ARMY RESEARCH LABORATORY

Performing XML Data Validation in the Global Force Management Data Initiative

by Frederick S. Brundick

ARL-TR-4742 **March 2009**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005

ARL-TR-4742 **March 2009**

Performing XML Data Validation in the Global Force Management Data Initiative

Frederick S. Brundick
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
March 2009	Final	October 2006 to December 2008

4. TITLE AND SUBTITLE
Performing XML Data Validation in the Global Force Management Data Initiative

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Frederick S. Brundick

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRD-ARL-CI-IC
Aberdeen Proving Ground, MD 21005

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-4742

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes the validation performed on data produced by the Global Force Management (GFM) Data Initiative (DI) project. Extensible Markup Language (XML) was chosen for the data exchange protocol because of its popularity and widespread support. The GFM data model, written in an object-oriented form, not only normalizes data, but also defines objects with parent/child hierarchical relationships. Since XML is a hierarchical language, the GFM XML schema is able to perform a more thorough analysis of hierarchical data than the same data presented in relational form. Due to the limitations of XSD tests, I wrote XML Stylesheet Language: Transformations (XSLT) scripts to perform additional structural and business rule validations on GFM XML data. This report contains descriptions and sample code from all of the GFM XSD modules. After an introduction to data validation with XSLT, the tests performed by both scripts are shown and explained. I present instructions on how to validate GFM XML data, along with sample results, and discuss the strengths and shortcomings of the validation process.

15. SUBJECT TERMS
XML, XSL, data schema, data validation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Frederick S. Brundick
Unclassified	Unclassified	Unclassified	UU	82	19b. TELEPHONE NUMBER (*Include area code*) 410-278-8943

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Holly Ingham, a fellow member of the Global Force Management Data Initiative team, and William Tanenbaum, a high school student in the Science and Engineering Apprentice Program, for reading early drafts of this report. They provided valuable feedback and kept the technical discussions from being overly esoteric.

INTENTIONALLY LEFT BLANK.

Executive Summary

This report describes the validation performed on data produced by the Global Force Management (GFM) Data Initiative (DI) project. Extensible Markup Language (XML) was chosen for the data exchange protocol because of its popularity and widespread support.

It is common practice to compare a set of XML data against an XML Schema definitio (XSD) file similar to the way that data in a relational database may be verifie against the database schema. While this process will catch gross errors, such as strings that are too long, incorrect date formats, and the absence of mandatory elements, much more validation may be performed. The two basic test classes that this report describes are structural and business rule validation.

One reason why XML is popular is because the values are surrounded by their element names, eliminating confusion caused by changing the order of unlabeled values. However, the GFM data model was written in an object-oriented form where the data is not only normalized (to use the database term), but also define objects with parent/child hierarchical relationships. XML is a hierarchical language, and this fact may be used to explicitly structure the data to match the hierarchical model.

The GFM XML schema is able to perform a more thorough analysis of hierarchical data than the same data presented in relational form. Tests are also define to check the data for referential integrity; when one object refers to a second object, the latter is checked to make sure that it exists. The fina structural test ensures that top-level identifier are unique.

There are limits to the tests that may be conducted by XSD. While the XML Stylesheet Language: Transformations (XSLT) was written to transform an XML fil into another form, such as Hypertext Markup Language (HTML), it may be used as a programming language to validate XML data. An XSLT script has been written to perform additional structural validations on GFM XML data. Unlike XSD, XSLT includes conditional expressions, allowing validation tests such as "the starting date/time group must be less than the termination date/time group." A second XSLT script enforces certain GFM business rules. While the XSD ensures that an particular value is in the set of values recognized by the data model, GFM business rules may put additional constraints on the value.

This report contains descriptions and sample code from all of the GFM XSD modules. After an introduction to data validation with XSLT, the tests performed by both scripts are shown and explained. I present instructions on how to validate GFM XML data, along with sample results, and discuss the strengths and shortcomings of the validation process.

While the intended audience is people with some XML experience, a glossary with acronyms is provided to defin terms in the context of the report. Appendix A contains a description of the Intelligence Community Information Security Marking (IC-ISM) XSD module, appendix B lists summaries of the structural and business rules, and appendix C describes a small set of sample data. This is work in progress that is expected to grow over time.

INTENTIONALLY LEFT BLANK.

1. Introduction

The Global Force Management (GFM) Data Initiative (DI) (*1*) uses Extensible Markup Language (XML) (*2*) to exchange data between servers and clients. It is important that the data be validated, not only to prevent retransmissions due to errors, but also to verify that the original data was constructed in accordance with the GFM schema, structure, and business rules. This report discusses the techniques that are used by GFM to ensure the correctness of the data.

An XML data fil is composed of elements (with attributes) in a hierarchical structure, while the original (non-XML) schema was written using a relational database model. Database tables contain records; each record is made up of fields with one value in each field The vocabulary in this report is a mixture of XML and database terms. It is simpler to call an element a table instead of using the verbose phrase "the XML element that corresponds to a table in the database model."[1] The intended meaning of a word should be clear within its context.

2. XML and Schemas

2.1 Background

While XML was originally designed to be used in conjunction with Hypertext Markup Language (HTML) and for document processing, it was quickly adopted as a data markup language (*3*). XML Schema Definitio (*4,5*) (commonly known as XML Schema or XSD) was adopted as a grammar to describe the structure and data types of an XML data file It is similar to a traditional data dictionary, declaring data elements and their types, but allows the programmer to defin his own data types, the structural relationships between elements, and the logical relationships.

There are two tests that may be performed on an XML data file The firs is *well-formedness*, which ensures that the fil is syntactically correct—are elements nested properly, are all elements terminated, are strings properly delimited, etc.? The second is *validation*, which compares the fil with an XSD and examines the data contents (*6*)—are all of the elements defined does all data conform to the type restrictions, are mandatory elements present, etc.?

Section 3 presents another methodology to validate data.

2.2 The GFM XSD

The GFM Information Exchange Data Model (IEDM) is an augmented subset of the Joint Command, Control and Consultation Information Exchange Data Model (JC3IEDM) developed

[1] See section 2.2.2 for other element mappings.

by the Multilateral Interoperability Programme Data Modelling Working Group (MIP DMWG) (*7*). The JC3IEDM is available in both relational and object-oriented XSDs. Note: All field and tables added by GFM start with "GFM_".

For readers who are unfamiliar with the GFMIEDM, a short explanation is in order. Every record of every table has a primary key called an Enterprise-Wide Identifie (EwID) (*8*). These surrogate keys are constructed so they are unique not only within a specifi table or database, but throughout all systems that produce GFM data. Because the JC3IEDM is an object-oriented model, even though it is intended to be used by relational database management systems (RDBMSs), many of the EwIDs are foreign keys that refer to records in other tables. The JC3IEDM may be treated as a highly normalized database where many of the tables have a parent-child relationship. A hierarchical (object-oriented) portion of the GFMIEDM Entity-Relationship (E-R) diagram, where links represent "is-a" relationships, is shown in figur 1.

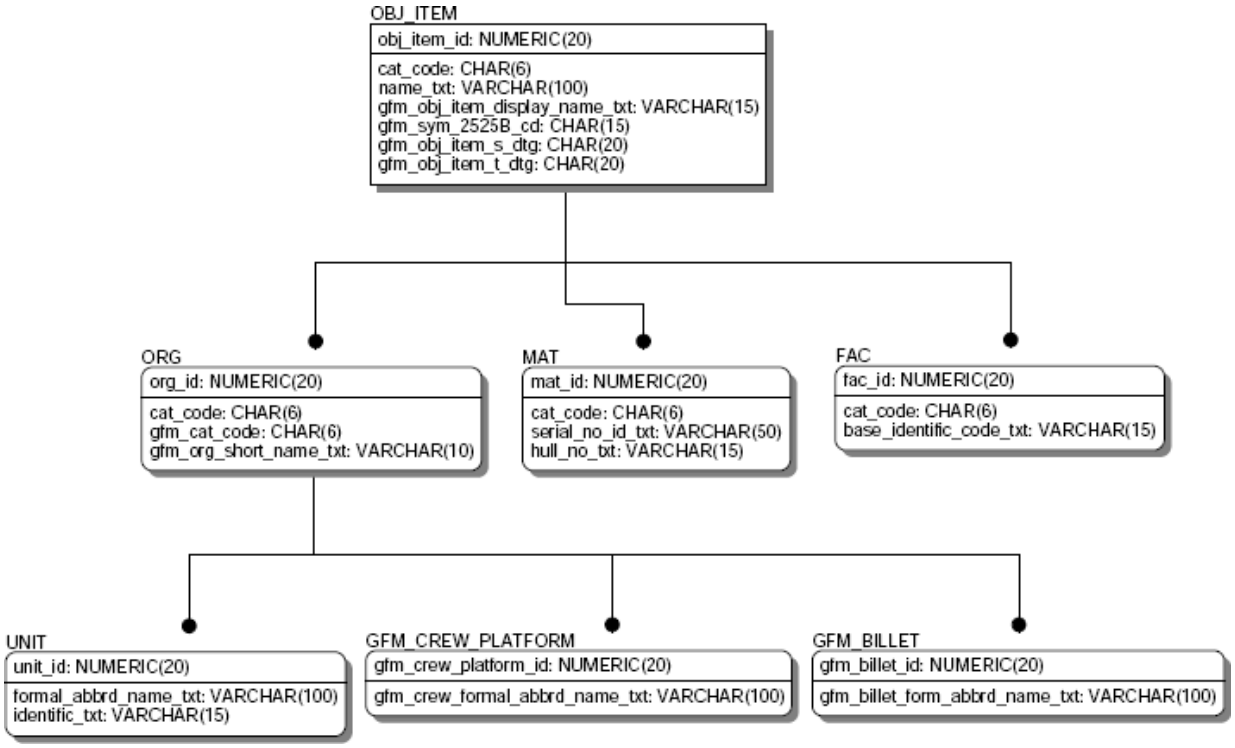

Figure 1. Hierarchical data schema.

The fiel "above the line" is the primary key for each table. In this example, **obj_item_id** is a unique key and the other EwIDs (such as **org_id** and **unit_id**) are foreign keys.

The relational XSD uses what the Oracle Corporation refers to as the "canonical XML format" (*9*), also called the "wrapper" format. Each database table is an XML element, the rows in the table correspond to children of the element, and the field are child elements of the row element. A typical example is shown in figur 2.[2] When hierarchical data is stored in canonical form, the relationships between the records is lost and validation becomes more difficult

[2]Intelligence Community Information Security Marking (IC-ISM) attributes are not shown but are described in appendix A.

```
<OBJ_ITEM_ASSOC_TBL>
  <OBJ_ITEM_ASSOC>
    <SUBJ_OBJ_ITEM_ID>72060755133858488</SUBJ_OBJ_ITEM_ID>
    <OBJ_OBJ_ITEM_ID>72060755133863257</OBJ_OBJ_ITEM_ID>
    <OBJ_ITEM_ASSOC_IX>72060755133858493</OBJ_ITEM_ASSOC_IX>
    <CAT_CODE>HSADMI</CAT_CODE>
    <SUBCAT_CODE>ALTFOR</SUBCAT_CODE>
    <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
    <GFM_SUBCAT_CODE>DEFALT</GFM_SUBCAT_CODE>
    <GFM_OBJ_ITEM_ASSOC_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_S_DTG>
    <GFM_OBJ_ITEM_ASSOC_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_T_DTG>
  </OBJ_ITEM_ASSOC>
  <OBJ_ITEM_ASSOC>
    <SUBJ_OBJ_ITEM_ID>72060755133858488</SUBJ_OBJ_ITEM_ID>
    <OBJ_OBJ_ITEM_ID>72060755133863255</OBJ_OBJ_ITEM_ID>
    <OBJ_ITEM_ASSOC_IX>72060755133858494</OBJ_ITEM_ASSOC_IX>
    <CAT_CODE>HSADMI</CAT_CODE>
    <SUBCAT_CODE>ALTFOR</SUBCAT_CODE>
    <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
    <GFM_SUBCAT_CODE>DEFALT</GFM_SUBCAT_CODE>
    <GFM_OBJ_ITEM_ASSOC_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_S_DTG>
    <GFM_OBJ_ITEM_ASSOC_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_T_DTG>
  </OBJ_ITEM_ASSOC>
  <OBJ_ITEM_ASSOC>
    <SUBJ_OBJ_ITEM_ID>72060755133858485</SUBJ_OBJ_ITEM_ID>
    <OBJ_OBJ_ITEM_ID>72060755133858483</OBJ_OBJ_ITEM_ID>
    <OBJ_ITEM_ASSOC_IX>72060755133858495</OBJ_ITEM_ASSOC_IX>
    <CAT_CODE>HSADMI</CAT_CODE>
    <SUBCAT_CODE>ALTFOR</SUBCAT_CODE>
    <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
    <GFM_SUBCAT_CODE>DEFALT</GFM_SUBCAT_CODE>
    <GFM_OBJ_ITEM_ASSOC_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_S_DTG>
    <GFM_OBJ_ITEM_ASSOC_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_T_DTG>
  </OBJ_ITEM_ASSOC>
</OBJ_ITEM_ASSOC_TBL>
```

Figure 2. Canonical (wrapper) data format.

Since XML is inherently hierarchical, it is ideally suited to handle GFM data. As figur 1 shows, a UNIT is-an ORG is-an OBJ_ITEM. To phrase that another way, a Unit is composed of an OBJ_ITEM, an ORG, and a UNIT. The **org_id** and **unit_id** field are foreign keys that have the same value as the **obj_item_id** primary key. It is incorrect to have a record in one of these tables without having a corresponding record in each of the other tables. Of course, that is just one path through the tree. A UNIT child of an ORG could be replaced by a GFM_CREW_PLATFORM or a GFM_BILLET, or the OBJ_ITEM child could be a MAT (Materiel) or FAC (Facility). These requirements—an OBJ_ITEM must have an ORG, MAT, or FAC child, and an ORG must have one of three children—may be explicitly recorded in the XSD.

Due to the size of the JC3IEDM, there are two separate sets of XSD files Since the GFM model is considerably smaller than the JC3 model, the relational and object-oriented schemas were combined. In order to improve readability and facilitate configuratio control, the XSD was split into multiple files These files which are shown in figur 3, are described from the bottom up.

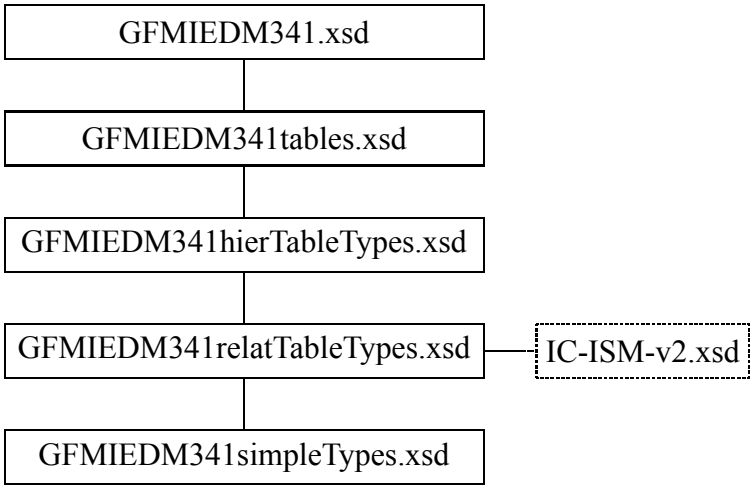

Figure 3. Files comprising the GFM XSD.

2.2.1 GFM Simple Data Types

The fil GFMIEDM341simpleTypes.xsd contains all of the data types that are define in the E-R model, translated into XML terms. The majority of the types were copied from the JC3IEDM XSD. At over 10,500 lines, this fil makes a strong argument for a modular design. There are 2 main classes of data types:

1. Simple or primitive types, and
2. Enumerated types.

An example of a simple type is shown in figur 4 and should be self-explanatory.

```
<xs:simpleType name="txt_mandatory_100">
  <xs:annotation>
    <xs:documentation>Oracle datatype: VARCHAR(100)</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:maxLength value="100"/>
  </xs:restriction>
</xs:simpleType>
```

Figure 4. Simple data type.

The word “mandatory” in the type name is slightly misleading. A *type* may not be mandatory; only an *element* may be mandatory.[3] This is a reminder that an element of this type is required. This type is an extension of **xs:string** and has a maximum length of 100 characters.[4] The ‘documentation’ element shows the equivalent SQL type.[5]

An enumerated type is an extension of **xs:string** with a list of all possible values. The example in figur 5 was chosen because it has only two values. Most of the enumerations are fairly large, and some are huge. The documentations and type names are taken from the E-R model. All JC3IEDM enumerated type names start with “DS”, while GFM types begin with “GF”.

```
<xs:simpleType name="DS4190_obj_type_estab_cat_code">
  <xs:annotation>
    <xs:documentation>Data type for the validation rule
        DS4190_obj_type_estab_cat_code</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string">
    <xs:enumeration value="CES">
      <xs:annotation>
        <xs:documentation>Complete equipment schedule: A list of the
            associated ancillaries, accessories, tools, literature and
            spares which, when scheduled together, form a composite
            vehicle, equipment or store.</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
    <xs:enumeration value="PCG">
      <xs:annotation>
        <xs:documentation>Parts catalogue: A list showing the disassembly
            build order of an equipment, identifying the assemblies,
            sub-assemblies and components which comprise the equipment
            (or assemblies and sub-assemblies).</xs:documentation>
      </xs:annotation>
    </xs:enumeration>
  </xs:restriction>
</xs:simpleType>
```

Figure 5. Enumerated data type.

2.2.2 GFM Relational Table Types

The fil GFMIEDM341relatTableTypes.xsd includes the fil GFMIEDM341simpleTypes.xsd. At roughly 1800 lines, it is the second largest file It define all of the “records” and “fields for the canonical form. It also imports the fil IC-ISM-v2.xsd (*10*) and define the attributes required for classificatio markings on data records.[6] These attributes are discussed in appendix A.

[3]The definitio is identical to the type **txt_optional_100**.

[4]All XML Schema elements are in the “xs” namespace and thus start with “xs:”.

[5]The XSD best practices recommends ‘documentation’ elements rather than comments because they are inline elements and therefore part of the schema.

[6]The ‘include’ element treats all of the declarations as part of a single application, while ‘import’ references declarations in another namespace.

Figure 6 shows the definitio of the **CivilianPostType** type. It is a complex type because it combines several elements together. By default, an 'all' element requires that each child element appears exactly once (in the data file) The firs element is named **CIV_POST_TYPE_ID** and it is of type **identifier2** (an EwID). Notice that the other two elements have almost the same type names except for the firs two characters. This was done to emphasize that the types are related, but that the firs was define by the JC3IEDM and the second by GFM.

```
<xs:complexType name="CivilianPostType">
  <xs:annotation>
    <xs:documentation>Definition: An ORGANISATION-TYPE with a set of
        duties that are intended to be fulfilled by one person in private
        sector and non-military government organisations.</xs:documentation>
  </xs:annotation>
  <xs:all>
    <xs:element name="CIV_POST_TYPE_ID" type="identifier20">
      <xs:annotation>
        <xs:documentation>Definition: The organisation-type-id of a
            specific CIVILIAN-POST-TYPE (a role name for object-type-id).
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="CAT_CODE" type="DS369_civ_post_type_cat_code">
      <xs:annotation>
        <xs:documentation>Definition: The specific value that represents
            the class of CIVILIAN-POST-TYPE.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="GFM_CAT_CODE" type="GF369_civ_post_type_gfm_cat_code">
      <xs:annotation>
        <xs:documentation>Definition: The specific value that represents
            the GFM class of CIVILIAN-POST-TYPE.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:all>
</xs:complexType>
```

Figure 6. Complex type.

The code fragment in figur 7 shows how to mark a fiel as optional. The attribute is 'minOccurs' and the value `0` means that the element may appear zero times, i.e., it is optional. Remember that the word "optional" in the type name is just a reminder to the human reader.

```
<xs:element name="DESCR_TXT" type="txt_optional_50" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Definition: The character string assigned to
        represent the specific ORGANISATION-TYPE.</xs:documentation>
  </xs:annotation>
</xs:element>
```

Figure 7. Optional element.

2.2.3 GFM Hierarchical Table Types

The file GFMIEDM341relatTableTypes.xsd could be included to define the schema using the canonical form; however, better validation may be achieved by taking advantage of the hierarchical nature of XML and defining hierarchical table types in the file GFMIEDM341hierTableTypes.xsd. To differentiate the force structure tree hierarchical data[7] from the data structure hierarchy, the latter is referred to as the *Generalization Hierarchy*. Nesting the elements also makes the data more understandable to a human reader since related data are kept together instead of being scattered throughout the GFM XML data file

Figure 8 shows part of the definition of **ObjectItem**. Notice that it uses 'sequence' instead of 'all'. The main difference, which is not relevant in GFM, is that the data elements in a sequence must appear in the specified order, while 'all' allows any order to be used. The reason that 'sequence' is used is to allow the ObjectItem type to be extended to derive a new type. The italicized element is explained in appendix A.

```
<xs:complexType name="ObjectItem">
  <xs:annotation>
    <xs:documentation>Definition: An individually identified object
    that has military or civilian significance.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="OBJ_ITEM_ID" type="identifier20">
      <xs:annotation>
        <xs:documentation>Definition: The unique value, or set of
        characters, assigned to represent a specific OBJECT-ITEM and to
        distinguish it from all other OBJECT-ITEMs.</xs:documentation>
      </xs:annotation>
    </xs:element>
    ...
  </xs:sequence>
  <xs:attributeGroup ref="SecurityAttributesGroup"/>
</xs:complexType>
```

Figure 8. Object Item type.

The simple types shown in figure 4 and 5 are *restriction*s. They are both based on the **xs:string** type, but the first has a maximum length of 100 characters and the second may contain only specified values.

A specification is needed to state that an ObjectItem must have a FAC, MAT, or ORG child element via an *extension* of the ObjectItem type. If one or more simple types were being added to the ObjectItem type, a format similar to the one for restricting simple types could be used; however, a *group* must be used instead. The group definition is shown in figure 9.

[7]Force structures are discussed in section 5.2 and appendix B. Otherwise, the word "hierarchy" refers to a data structure.

```
<xs:group name="ObjectItemGroup">
  <xs:choice>
    <xs:element name="FAC" type="Facility"/>
    <xs:element name="MAT" type="Materiel"/>
    <xs:element name="ORG" type="OrganisationHierarchy"/>
  </xs:choice>
</xs:group>
```

Figure 9. Object Item Group.

The 'choice' element specifies that exactly one of the elements listed must appear in the data. Now the parts may be assembled to define the hierarchical version of an Object Item type. This has been done in figure 10. An ObjectItemHierarchy consists of all of the elements in an ObjectItem (figure 8) plus the ObjectItemGroup (figure 9).

```
<xs:complexType name="ObjectItemHierarchy">
  <xs:annotation>
    <xs:documentation>An ObjectItem which is part of a hierarchy.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ObjectItem">
      <xs:sequence>
        <xs:group ref="ObjectItemGroup"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 10. Object Item Hierarchy type.

Figure 9 states that the element **ORG** is of type **OrganisationHierarchy**. This hierarchical type is an extension of the **Organisation** type plus the OrganisationGroup. Except for the names, these types and this group are defined the same way as the Object Item example. The final result matches the hierarchical data schema shown in figure 1.

2.2.4 GFM Table Elements

All of the pieces are now in place to define the "table" elements. The file GFMIEDM341tables.xsd includes the file GFMIEDM341hierTableTypes.xsd. Next a type for the root element is defined to allow different main XSD files to include this file. As explained in the next section, validation rules must appear in the scope of the element, so the top element may not be declared here. The enclosing elements and some table elements are shown in figure 11.

The first thing to note is that every table is optional. If a GFM XML data file does not contain any MiscellaneousEquipmentType data, then it should not have a MISC_EQPT_TYPE_TBL element.[8]

[8] A basic tenet of XML data is that empty elements are rarely used.

```
<xs:complexType name="GFMIEDM34Type">
  <xs:annotation>
    <xs:documentation xml:lang="en">This element MUST be conveyed as
    the root element in any instance document based on this schema
    specification.</xs:documentation>
  </xs:annotation>
  <xs:all>
    ...
    <xs:element ref="MISC_EQPT_TYPE_TBL" minOccurs="0"/><!--rel-->
    <xs:element ref="OBJ_ITEM_TBL" minOccurs="0"/><!--rel-->
    <xs:element ref="OBJ_ITEM_OO_TBL" minOccurs="0"/><!--oo-->
    <xs:element ref="OBJ_ITEM_ADDR_TBL" minOccurs="0"/><!--both-->
    ...
  </xs:all>
</xs:complexType>
```

Figure 11. Root (main) type.

Similarly, a table may appear no more than once. Also, all of the elements end with "_TBL", a convention adopted by the JC3IEDM.

Finally, compare the two highlighted elements. The firs is the relational (canonical) Object Item table, and the second is the object-oriented (generalization hierarchy) version. This fact is emphasized by the `<!--rel-->` and `<!--oo-->` comments. Within a GFM XML data file a hierarchical table element ends with "_OO_TBL". The fourth element shown has a `<!--both-->` comment to denote that it is not part of any generalization hierarchy, and thus is the same regardless of the schema being used.

The GFM XSD define 46 tables; 37 are part of generalization hierarchies and 9 are purely relational. The hierarchical tables are combined into only three XML hierarchies.

The rest of the fil define all of the table elements and states that each element may contain one or more child elements (records). (A sequence implicitly sets 'minOccurs=1'.) A simple table is shown in figur 12. All relational tables use this form, changing only the element and type names.

```
<xs:element name="OBJ_ITEM_TBL"><!--rel-->
  <xs:annotation><xs:documentation>rel only</xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OBJ_ITEM" type="ObjectItem"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 12. Simple table definition

The hierarchical tables, such as the one in figur 13, start out like the relational tables. The element name ends with "_OO_TBL" instead of "_TBL", but the "record" name remains the same. The **OBJ_ITEM_TBL** contains one or more **OBJ_ITEM**s, while the **OBJ_ITEM_OO_TBL** contains one or more top-level **OBJ_ITEM**s, which have child elements.

What complicates the definitio of a hierarchical table are the 'key' and 'keyref' elements, which are highlighted in figur 13. Each of these elements has a name, a selector with an XPath[9] expression, and a fiel with a second XPath expression (*11*). In addition, a keyref contains a reference to a key.

```
<xs:element name="OBJ_ITEM_OO_TBL"><!--oo-->
  <xs:annotation><xs:documentation>oo only</xs:documentation></xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="OBJ_ITEM" type="ObjectItemHierarchy"
                  maxOccurs="unbounded"/>
        <xs:annotation><xs:documentation>foreign keys in OBJ_ITEM tree must
        equal primary key</xs:documentation></xs:annotation>
        <xs:key name="ObjItemIDTree">
          <xs:selector xpath="OBJ_ITEM_ID"/>
          <xs:field xpath="."/>
        </xs:key>
        <xs:keyref name="testFacID" refer="ObjItemIDTree">
          <xs:selector xpath="FAC"/>
          <xs:field xpath="FAC_ID"/>
        </xs:keyref>
        <xs:keyref name="testMatID" refer="ObjItemIDTree">
          <xs:selector xpath="MAT"/>
          <xs:field xpath="MAT_ID"/>
        </xs:keyref>
        <xs:keyref name="testOrgID" refer="ObjItemIDTree">
          <xs:selector xpath="ORG"/>
          <xs:field xpath="ORG_ID"/>
        </xs:keyref>
        <xs:keyref name="testGFMBilletID" refer="ObjItemIDTree">
          <xs:selector xpath="ORG/GFM_BILLET"/>
          <xs:field xpath="GFM_BILLET_ID"/>
        </xs:keyref>
        <xs:keyref name="testGFMCrewPlatformID" refer="ObjItemIDTree">
          <xs:selector xpath="ORG/GFM_CREW_PLATFORM"/>
          <xs:field xpath="GFM_CREW_PLATFORM_ID"/>
        </xs:keyref>
        <xs:keyref name="testUnitID" refer="ObjItemIDTree">
          <xs:selector xpath="ORG/UNIT"/>
          <xs:field xpath="UNIT_ID"/>
        </xs:keyref>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 13. Hierarchical table definitio with tests.

[9]XPath is a standard XML query language. XSD uses a subset of XPath.

The purpose of key/keyref is to test referential integrity. The ‘selector’ and ‘field are used to locate an element whose value is associated with the key’s name. In this example, the selector specifie the **OBJ_ITEM_ID**. Since we have located the element that we want, the fiel is “.”, which means the current element. A simple way to picture this is the assignment statement

ObjItemIDTree = value-of(this OBJ_ITEM’s OBJ_ITEM_ID).

The ‘keyref’s that follow specify elements whose values must be the same as a key. The highlighted keyref has a name that must be unique but otherwise is ignored. The ‘refer’ attribute supplies the name of the desired key. The ‘selector’ and ‘field attributes specify that the **ORG_ID** of the **ORG** child of the current **OBJ_ITEM** is the element to compare. The equivalent comparison statement is

value-of(this OBJ_ITEM’s ORG’s ORG_ID) == ObjItemIDTree.

These statements may be combined into the desired result, namely, when an OBJ_ITEM has a child ORG, the ORG’s ORG_ID must be equal to the OBJ_ITEM’s OBJ_ITEM_ID. Since this test is applied to every **OBJ_ITEM** in the **OBJ_ITEM_OO_TBL**, it roughly the same as the SQL query shown in figur 14. The value returned by the query should match the number of records in the ORG table.

```
SELECT COUNT(*)
  FROM OBJ_ITEM oi, ORG o
  WHERE oi.obj_item_id=o.org_id;
```

Figure 14. Equivalent SQL referential test.

2.2.5 Main GFM XSD File

The GFMIEDM341.xsd fil is the main GFM XSD fil and is entirely devoted to validation tests. If fewer restrictions are desired, such as for GFM XML data file that are known to be incomplete, then a different main fil could be written. The reason for the declaration of the GFMIEDM34Type should now be apparent, since tests must be define within the scope (body) of an element.

After including the GFMIEDM341tables.xsd file the main fil define the document root of a GFM XML data file namely **GFMIEDM34**. The firs rule is verbose but easy to understand—it uses the ‘unique’ element to ensure that the primary key in each record (that is not a child in a generalization hierarchy) is unique. A highly edited version of the uniqueness test is shown in figur 15. The selector’s XPath expression, of which only two paths are shown, must be confine to a single, long line. The actual test lists all 12 EwIDs that appear in the GFM XSD.

```
<xs:unique name="UniqueEwID">
  <xs:annotation><xs:documentation>primary EwIDs must be unique
  </xs:documentation></xs:annotation>
    <xs:selector xpath="
        ...|*/OBJ_ITEM/OBJ_ITEM_ID|
        OBJ_ITEM_ADDR_TBL/OBJ_ITEM_ADDR/OBJ_ITEM_ADDR_IX|..."
    />
  <xs:field xpath="."/>
</xs:unique>
```

Figure 15. Uniqueness test.

The paths in the selector expression are separated by vertical bars, and the field s XPath is simply ".". Ignoring the syntactic details, what this test indicates is that all of the field listed in the selector must contain unique values. Each path must be long enough to unambiguously specify a single element. The foreign key element often uses the same name as the primary key element, requiring the use of long paths to differentiate between the two. The example shows that the path always starts with the "_TBL" name and drills down to the EwID element. The firs path in the example starts with a wildcard (*) because an **OBJ_ITEM** may be a record in an **OBJ_ITEM_TBL** or an **OBJ_ITEM_OO_TBL**.[10]

The rest of the fil consists of tests for referential integrity. There are two types of references: pointers to disparate element types ("horizontal" references) and the relational version of the data hierarchy ("vertical" references). The keys are define the same way as in the GFMIEDM341tables.xsd file Each key has the same name as its field plus vertical keys have the prefi "Rel".

The details are explained with the OBJ_ITEM hierarchy. Figure 16 shows the keys that are define for field in the **OBJ_ITEM**. The firs two selectors start with a wildcard, because horizontal tests apply to both relational and object-oriented tables. The third selector explicitly names the table element because the key is used only in vertical relational tests.

```
<!-- ObjItem keys -->
<xs:key name="ObjItemID">
  <xs:selector xpath="*/OBJ_ITEM"/>
  <xs:field xpath="OBJ_ITEM_ID"/>
</xs:key>
<xs:key name="OrgID">
  <xs:selector xpath="*/OBJ_ITEM/ORG"/>
  <xs:field xpath="ORG_ID"/>
</xs:key>
<xs:key name="RelOrgID">
  <xs:selector xpath="ORG_TBL/ORG"/>
  <xs:field xpath="ORG_ID"/>
</xs:key>
```

Figure 16. Object Item keys.

[10]Remember that these paths are relative to the **GFMIEDM34** element.

The horizontal tests in figur 17 are for the table (OBJ_ITEM_ALIAS) that associates an Object Item with an Alias. The desired tests are to make sure that the elements that are referenced, namely the **OBJ_ITEM_ID** and **GFM_OBJ_ALIAS_TYPE**, exist in the GFM XML data file The firs keyref ensures that the OBJ_ITEM_ID in an OBJ_ITEM_ALIAS of an OBJ_ITEM_ALIAS_TBL has the same value as an ObjItemID as define in figur 16. The second does the same for the Alias end of the link; the key definitio is not shown.

```
<!-- ObjItemAddr -->
<xs:keyref name="ObjItemAliasObjItemRef" refer="ObjItemID">
  <xs:selector xpath="OBJ_ITEM_ALIAS_TBL/OBJ_ITEM_ALIAS"/>
  <xs:field xpath="OBJ_ITEM_ID"/>
</xs:keyref>
<xs:keyref name="ObjItemAliasAliasTypeRef" refer="GFMAliasTypeID">
  <xs:selector xpath="OBJ_ITEM_ALIAS_TBL/OBJ_ITEM_ALIAS"/>
  <xs:field xpath="GFM_OBJI_ALIAS_TYPE"/>
</xs:keyref>
```

Figure 17. Object Item Alias keyrefs.

The keyrefs in figur 18 match the E-R diagram in figur 1. Each EwID in a table must match the EwID in its parent table. However, there are subtle differences between these tests and the ones define in figur 13. For one thing, the scope is entirely different. The other tests are define inside of the **OBJ_ITEM_OO_TBL**, while these keys and keyrefs are in the main **GFMIEDM34** element.

Another difference is the key that is referenced—the object-oriented keyrefs all refer to the **OBJ_ITEM_ID** element. The **OBJ_ITEM_OO_TBL** element requires that each OBJ_ITEM has an ORG, MAT, or FAC child, and an ORG has one of three child elements. Since the relational elements are distinct from each other, this restriction must be imposed by the keyrefs from the bottom up. A **UNIT_ID** must match an **ORG_ID**, while an **ORG_ID** must match an **OBJ_ITEM_ID**. It is still possible to create an ORG without a child, but detecting this is beyond the scope of XSD. The techniques described in the next section could perform this task, but the emphasis has been on GFM XML data in the object-oriented form.

3. Validating Data With Transformations

3.1 Background

There are limits to the tests that may be performed by an XSD file Only a subset of XPath is recognized, and conditional statements are not allowed. A separate application must be written to conduct additional validation tests, and that can be quite an undertaking. Rick Jelliffe of the Academia Sinica devised a way to use Extensible Stylesheet Language: Transformations (XSLT) (*12*) as a clever alternative.

```
<!-- ObjItem tree -->
<xs:keyref name="OrgRef" refer="ObjItemID">
  <xs:selector xpath="ORG_TBL/ORG"/>
  <xs:field xpath="ORG_ID"/>
</xs:keyref>
<!-- Org subtree -->
<xs:keyref name="UnitRef" refer="RelOrgID">
  <xs:selector xpath="UNIT_TBL/UNIT"/>
  <xs:field xpath="UNIT_ID"/>
</xs:keyref>
<xs:keyref name="CrewPlatformRef" refer="RelOrgID">
  <xs:selector xpath="GFM_CREW_PLATFORM_TBL/GFM_CREW_PLATFORM"/>
  <xs:field xpath="GFM_CREW_PLATFORM_ID"/>
</xs:keyref>
<xs:keyref name="BilletRef" refer="RelOrgID">
  <xs:selector xpath="GFM_BILLET_TBL/GFM_BILLET"/>
  <xs:field xpath="GFM_BILLET_ID"/>
</xs:keyref>
<!-- Mat subtree -->
<xs:keyref name="MatRef" refer="ObjItemID">
  <xs:selector xpath="MAT_TBL/MAT"/>
  <xs:field xpath="MAT_ID"/>
</xs:keyref>
<!-- Fac subtree -->
<xs:keyref name="FacRef" refer="ObjItemID">
  <xs:selector xpath="FAC_TBL/FAC"/>
  <xs:field xpath="FAC_ID"/>
</xs:keyref>
```

Figure 18. Relational data hierarchy keyrefs.

XSLT was originally conceived to transform one XML document into another (*13*). The fil GFMIEDM341flatten.xsl is an XSLT script that reads a GFM XML data fil that contains hierarchical data and produces the same data in relational form. The most common use of XSLT is to indicate to a Web browser how to convert XML data into XHTML (*14*) (a reformulation of HTML) to make it more readable by humans.[11] GFM's script to do this is the fil GFMIEDM341.xsl. Jelliffe describes how to write XSLT templates (rules) to produce error messages in XHTML based on problems that it detects in the XML data (*15*).

The GFM package contains two XSLT scripts. These scripts assume that the data has already been validated against the GFM XSD. The first GFMIEDM341validate.xsl, define structural restrictions that could not be tested by the XSD. The second, GFMIEDM341businessRules.xsl, is an attempt to automate business rules that are not formally part of the model. The restrictions and allowed values are summarized in appendix B. Other scripts may be written to validate data with special requirements. Providing tools to perform data validation allows the data producers to test their systems, while data consumers may avoid tedious input checking in their applications.

[11] All major Web browsers have a built-in XSLT transformation engine.

3.2 XSLT Validation Overview

This report is not intended to be a tutorial on XSLT or XPath. However, XSLT is a *declarative* language, not a *procedural* language like C, Java, and most other popular languages. An overview is provided to explain Jelliffe's technique in layman's terms, followed by actual code from the GFM XSLT files

An XSLT fil contains *templates*. Most templates contain a 'match' attribute, which consists of an XPath expression.[12] This is very similar to the patterns in an XSD key or keyref, except the full power of XPath is available. The 'apply-templates' element may be used to compare the XML data tree (or a specifie subtree) to all of the templates or just a subset. The code in a matching template is executed, possibly producing XHTML output.

Without going into too much detail, a *match* template may have an optional 'mode' and/or 'priority'. By default, all templates have no mode or priority. Each template in the desired mode is evaluated, comparing its XPath expression against the XML tree (or subtree). The one with the highest priority "wins" and its code is evaluated. A default template that matches every element and prints an element's contents is automatically provided to make sure that all data gets matched (and printed) and is never ignored.

The other kind of template is a *named* template, where a f xed name takes the place of an XPath expression. This is XSLT's concession to procedural code, allowing parameters to be passed to a specifi template to facilitate code re-use. These are used in the GFM scripts to format error messages.

The mechanism that makes Jelliffe's technique possible is the fact that once XSLT matches a data element against a template, other templates are ignored. A template that has no contents "traps" an element and quietly ignores it.

The template in figur 19[13] catches all text elements[14] when the *mode_name* mode is in effect and throws them away. The priority of `-1` causes the template to be matched after other templates.

```
<xsl:template match="text()" priority="-1" mode="mode_name">
  <!-- strip characters -->
</xsl:template>
```

Figure 19. Default empty template for mode *mode_name*.

Most templates used in GFM validation are "bad" templates. They match data that has an error and generate XHTML that describes the problem that was found. All elements that are not matched are processed by a default template like the one in figur 19. The assumption is that all elements that are not bad are either good or they are irrelevant.

The opposite idea is to write a template that matches "good" data and does nothing with it. A second template, with a similar 'match' expression but a lower priority, produces the proper error message. The default template then catches all data elements that are left and ignores them.

[12] A match template is analogous to an SQL query statement.

[13] By convention, XSLT tokens are in the "xsl" namespace and thus start with "xsl:".

[14] There are seven types of nodes in the XML Tree Model. All GFM data elements are text elements.

A variation on these two approaches is a template that matches a subset of data, then uses conditional processing to determine if the data is good or bad. The firs of these is the link test as described in the next section.

3.3 Structural Validation

3.3.1 Link Validation

The simplest template in the fil GFMIEDM341validate.xsl, because it is the closest to procedural code, is the template that detects Object Item Associations (Assocs) that are missing either the parent or child node. The actual template is shown in figur 20.[15]

```
<!-- locate nodes where an endpoint is not an OBJ_ITEM -->
<!-- external OBJ_ITEMs are not supported yet -->
<xsl:template priority="2"
  match="//gfm:OBJ_ITEM_ASSOC"
  mode="link">
  <xsl:choose>
    <xsl:when test="gfm:SUBJ_OBJ_ITEM_ID=//gfm:OBJ_ITEM/gfm:OBJ_ITEM_ID">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="missing-endpoint">
        <xsl:with-param name="class" select="'OBJ_ITEM_ASSOC'"/>
        <xsl:with-param name="end-name"  select="'SUBJ_OBJ_ITEM_ID'"/>
        <xsl:with-param name="end-ewid" select="gfm:SUBJ_OBJ_ITEM_ID"/>
        <xsl:with-param name="link-ewid" select="gfm:OBJ_ITEM_ASSOC_IX"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:choose>
    <xsl:when test="gfm:OBJ_OBJ_ITEM_ID=//gfm:OBJ_ITEM/gfm:OBJ_ITEM_ID">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="missing-endpoint">
        <xsl:with-param name="class"     select="'OBJ_ITEM_ASSOC'"/>
        <xsl:with-param name="end-name"  select="'OBJ_OBJ_ITEM_ID'"/>
        <xsl:with-param name="end-ewid"  select="gfm:OBJ_OBJ_ITEM_ID"/>
        <xsl:with-param name="link-ewid" select="gfm:OBJ_ITEM_ASSOC_IX"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 20. Template that checks for invalid OBJ_ITEM_ASSOC links.

The priority is 2 and the mode is `link`. The XPath expression is very simple. The code

```
match="//gfm:OBJ_ITEM_ASSOC"
```

[15] All elements define in the GFM XSD are in the "gfm" namespace.

will match all **OBJ_ITEM_ASSOC** elements at any depth in the XML data tree. The ‘choose’ is the outer wrapper for a multi-branch conditional statement. Each ‘when’ element may be thought of as an “if” or “else if”. The ‘otherwise’ element is a default (“else”) that is executed if none of the ‘when’ tests is true.

The ‘test’ attributes are XPath expressions. While they may look intimidating, GFM uses basic patterns. The data element that matches the ‘match’ expression may be treated as an argument to the template. All expressions within the template are relative to it. Therefore, the reference to

```
gfm:SUBJ_OBJ_ITEM_ID
```

means “the value of the SUBJ_OBJ_ITEM_ID in the current OBJ_ITEM_ASSOC.”

The second fragment uses an absolute path

```
//gfm:OBJ_ITEM/gfm:OBJ_ITEM_ID
```

and means “the value of the OBJ_ITEM_ID of any OBJ_ITEM” in the data. The “=” means “is equal to.”

The entire expression may be summed up as “For each OBJ_ITEM_ASSOC, is there an OBJ_ITEM whose OBJ_ITEM_ID is equal to the Assoc’s SUBJ_OBJ_ITEM_ID?” If so, do nothing (the ‘when’ element is empty), otherwise call the named template “missing-endpoint” with the listed parameters. The code is shown in figur 21.

```
<!-- parent or child of link does not exist -->
<xsl:template name="missing-endpoint">
  <xsl:param name="class"/>
  <xsl:param name="end-name"/>
  <xsl:param name="end-ewid"/>
  <xsl:param name="link-ewid"/>
  <li>
    <xsl:value-of select="$class"/>
    <xsl:text>’s␣</xsl:text>
    <xsl:value-of select="$end-name"/>
    <xsl:text>␣(</xsl:text>
    <xsl:value-of select="$end-ewid"/>
    <xsl:text>)␣does␣not␣exist</xsl:text>
    <br />
    <xsl:value-of select="$link-ewid"/>
  </li>
</xsl:template>
```

Figure 21. Named template “missing-endpoint” that produces output.

The named template starts with the parameter names (argument list); the rest is written to the output file It uses the XHTML tags ‘li’ and ‘br’. In this case, the ‘value-of’ element writes the value of the named parameter. The ‘text’ element writes literals that contain blanks, which are shown as “␣” symbols. Sample output is shown in section 4.2.

Going back to the original template, the firs two parameters in figur 20 are literal strings to facilitate code re-use, while the other two are values taken from the current OBJ_ITEM_ASSOC data element.

The second 'choose' element tests for the existence of the child OBJ_ITEM of the link. This template will detect a link whose parent, child, or both is missing from the GFM XML data file This may not be the case with other templates, which may stop after the firs error.

The endpoint tests performed by this template could have been done with a key/keyref pair, and in fact, those tests were originally in the GFM XSD. Links to external data stored in other systems will be added, and these tests will be embellished to check for external indicators.

The links between Object Types are more sophisticated, and their validation tests must be performed with XSLT. Figure 22 shows the firs part of the template. It ensures that the parent Object Type and Object Type Establishment both exist in the GFM XML data file Again, this may be a key/keyref in the XSD if external nodes are ignored.

There are three types of Object Type links (OBJ_TYPE_ESTAB_OBJT_DET or OTEOD) in the GFM model, and they are designated by the **GFM_OTEOD_ROLE_IND_CD** element. A normal OTEOD has a role of "0", and currently that's the only value used in GFM data. A Type 1 Role is a placeholder that shows that an Organisation must be created, but the details are unknown. This type of OTEOD should *not* have a child node. The third possible value for a role is "2", but it is ignored for now and is treated as if it were a Type 0 Role. These tests, which occur in the second half of the template, are shown in figur 23.

The outermost 'choose' element checks if the OTEOD is a Type 1 Role; if so, the template exits. Otherwise, the child Object Type and Object Type Establishment are checked to make sure that they exist. As it is currently written, the only difference between this template and a group of key/keyrefs is the test of the role indicator code.

The "text()" template with a mode of "link" and a priority of `-1` is required because of all of the data elements that are not Assocs or OTEODs. Without this default template, all other data elements would be printed.

```
<!-- locate nodes where an endpoint is not an OBJ_TYPE(_ESTAB) -->
<!-- exception: Type 1 Roles have a parent but not a child -->
<!-- external OBJ_TYPEs are not supported yet -->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE_ESTAB_OBJT_DET"
  mode="link">
  <xsl:choose>
    <xsl:when test="gfm:ESTABD_OBJ_TYPE_ID=//gfm:OBJ_TYPE/gfm:OBJ_TYPE_ID">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="missing-endpoint">
        <xsl:with-param name="class"     select="'OBJ_TYPE_ESTAB_OBJT_DET'"/>
        <xsl:with-param name="end-name"  select="'ESTABD_OBJ_TYPE_ID'"/>
        <xsl:with-param name="end-ewid"  select="gfm:ESTABD_OBJ_TYPE_ID"/>
        <xsl:with-param name="link-ewid"
                        select="gfm:OBJ_TYPE_ESTAB_OBJT_DET_IX"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:choose>
    <xsl:when test="gfm:OBJ_TYPE_ESTAB_IX=
                  //gfm:OBJ_TYPE_ESTAB/gfm:OBJ_TYPE_ESTAB_IX">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="missing-endpoint">
        <xsl:with-param name="class"     select="'OBJ_TYPE_ESTAB_OBJT_DET'"/>
        <xsl:with-param name="end-name"  select="'OBJ_TYPE_ESTAB_IX'"/>
        <xsl:with-param name="end-ewid"  select="gfm:OBJ_TYPE_ESTAB_IX"/>
        <xsl:with-param name="link-ewid"
                        select="gfm:OBJ_TYPE_ESTAB_OBJT_DET_IX"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
  ...
```

Figure 22. Template that checks for invalid OBJ_TYPE_ESTAB_OBJT_DET links, part 1.

```
  ...
  <xsl:choose>
    <xsl:when test="gfm:GFM_OTEOD_ROLE_IND_CD='1'">
    </xsl:when>
    <xsl:otherwise>
      <xsl:choose>
        <xsl:when test="gfm:DET_OBJ_TYPE_ID=//gfm:OBJ_TYPE/gfm:OBJ_TYPE_ID">
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="missing-endpoint">
            <xsl:with-param name="class"
                            select="'OBJ_TYPE_ESTAB_OBJT_DET'"/>
            <xsl:with-param name="end-name" select="'DET_OBJ_TYPE_ID'"/>
            <xsl:with-param name="end-ewid" select="gfm:DET_OBJ_TYPE_ID"/>
            <xsl:with-param name="link-ewid"
                            select="gfm:OBJ_TYPE_ESTAB_OBJT_DET_IX"/>
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
      <xsl:choose>
        <xsl:when test="gfm:DET_OBJ_TYPE_ESTAB_IX=
                        //gfm:OBJ_TYPE_ESTAB/gfm:OBJ_TYPE_ESTAB_IX">
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="missing-endpoint">
            <xsl:with-param name="class" select="'OBJ_TYPE_ESTAB_OBJT_DET'"/>
            <xsl:with-param name="end-name"
                            select="'DET_OBJ_TYPE_ESTAB_IX'"/>
            <xsl:with-param name="end-ewid"
                            select="gfm:DET_OBJ_TYPE_ESTAB_IX"/>
            <xsl:with-param name="link-ewid"
                            select="gfm:OBJ_TYPE_ESTAB_OBJT_DET_IX"/>
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 23. Template that checks for invalid OBJ_TYPE_ESTAB_OBJT_DET links, part 2.

3.3.2 Category Code Validation

Every object in the generalization hierarchy has a fiel named **CAT_CODE**, which states the category of its child (if any). The firs template in the XSLT fil checks every element in the generalization hierarchy to make sure that the CAT_CODE and child element agree. It does all of the work in the 'match' attribute instead of using a procedural conditional element. Since it is looking for bad patterns, the template find elements whose CAT_CODE is *not* the same as the child. One of the many templates is shown in figur 24.

```
<!-- locate nodes where child exists but catcode is incorrect -->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE[gfm:CAT_CODE!='OR'][gfm:ORG_TYPE]"
  mode="tree">
  <xsl:call-template name="cc-mismatch">
    <xsl:with-param name="class"    select="'OBJ_TYPE'"/>
    <xsl:with-param name="catcode"  select="gfm:CAT_CODE"/>
    <xsl:with-param name="subclass" select="'ORG_TYPE'"/>
    <xsl:with-param name="ewid"     select="gfm:OBJ_TYPE_ID"/>
    <xsl:with-param name="name"     select="gfm:NAME_TXT"/>
  </xsl:call-template>
</xsl:template>
```

Figure 24. Template that compares CAT_CODE with child element.

The XPath expression matches "an OBJ_TYPE that has an ORG_TYPE child but whose CAT_CODE element does not have the value `OR`." There are a total of 34 templates that test the combinations allowed by the model. If an element contains an error, the child elements are not tested because the transformation engine has already found a match.

3.3.3 Validation of Override Pairs

The GFM model adds enumerated values to existing JC3 fields To facilitate future data exchanges between GFM- and JC3-based systems, new field were define containing the new enumerated values. The implementation requirement states "If the GFM fiel is `NOS` (no such), then use the value in the JC3 field else use the GFM field s value." What this means in practice is that it is invalid for both field to have non-`NOS` values.[16] The template in figur 25 will be executed when an **ORG** is found where neither the **CAT_CODE** nor the **GFM_CAT_CODE** is `NOS`.

There are some JC3 field that do not include the `NOS` value. In these cases, the test uses the firs enumerated JC3 value instead. This value is relevant only when the GFM fiel is `NOS`, but the data should be consistent. There are 11 override templates.

[16] Setting both field to `NOS` is perfectly valid and means the desired value is `NOS`.

```
<!-- locate nodes where each field pair is not NOS -->
<xsl:template priority="2"
  match="//gfm:ORG[gfm:CAT_CODE!='NOS'][gfm:GFM_CAT_CODE!='NOS']"
  mode="override">
  <xsl:call-template name="no-override">
    <xsl:with-param name="class"     select="'ORG'"/>
    <xsl:with-param name="jc3-name"  select="'CAT_CODE'"/>
    <xsl:with-param name="jc3-value" select="gfm:CAT_CODE"/>
    <xsl:with-param name="gfm-name"  select="'GFM_CAT_CODE'"/>
    <xsl:with-param name="gfm-value" select="gfm:GFM_CAT_CODE"/>
    <xsl:with-param name="ewid"      select="gfm:ORG_ID"/>
  </xsl:call-template>
</xsl:template>
```

Figure 25. Template that find incorrectly overridden fields

3.3.4 Validation of Date/Time Groups

Many tables contain a pair of elements that defin the start and termination date/time group (DTG) of the object; the start DTG must be before (less than) the termination DTG. This would normally not be a difficul test, but XSLT cannot convert the string into an actual date.[17] The trick is to delete extraneous characters, convert each string into a number, and compare the values. This may be safely done because the DTGs have been validated against the XSD, and the type has a strict pattern that must be adhered to, which includes leading zeroes where needed.

The function **translate** is used to replace a set of characters with an empty string. This string is then converted into a number with the **number** function. To give an example, a typical DTG is "1990-01-01T00:00:00Z". Removing the dashes, colons, "T", and "Z" gives "19900101000000". Because the date is stored in year-month-day form, the numerical values may be compared. The template for a start and termination pair is shown in figur 26.

This is another example of a template that matches a set of elements, then uses a conditional (in this case, an 'if' element) to check for an invalid condition. There are 12 templates that test DTG ranges.

There are two field whose values must be within the start and termination DTG range. The valid expression is $s\_dtg \leq date\_time < t\_dtg$. Unfortunately, XPath 1.0 does not have a "$\leq$" operator, so the firs part of the test must be rewritten as "not $(s\_dtg > date\_time)$". The "$>$" and "$<$" and symbols are not allowed in a 'when' element and have been replaced by "`>`" and "`<`", respectively. The template that tests both the start and termination DTG range and the effective datetime is shown in figur 27.

[17] There may be extensions that do this, and newer versions of XML and XSLT may have this capability, but GFM has been restricted to the original versions because they are widely available.

```
<!-- locate objects where an S_DTG is not less than the T_DTG -->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE"
  mode="dtg">
  <xsl:if
    test="number(translate(gfm:GFM_OBJ_TYPE_S_DTG, '-T:Z', '')) >=
          number(translate(gfm:GFM_OBJ_TYPE_T_DTG, '-T:Z', ''))">
    <xsl:call-template name="bad-dtg-range">
      <xsl:with-param name="class" select="'OBJ_TYPE'"/>
      <xsl:with-param name="s-dtg" select="gfm:GFM_OBJ_TYPE_S_DTG"/>
      <xsl:with-param name="t-dtg" select="gfm:GFM_OBJ_TYPE_T_DTG"/>
      <xsl:with-param name="ewid"  select="gfm:OBJ_TYPE_ID"/>
      <xsl:with-param name="name"  select="gfm:NAME_TXT"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Figure 26. Template that find invalid DTG ranges.

The comment block at the top of the figur summarizes the actions performed. The **EFFCTV_DTTM** element is optional, and errors will be falsely reported if it is absent from the data file The expression `test="gfm:EFFCTV_DTTM"` is true if the **EFFCTV_DTTM** element has a value. The format for **EFFCTV_DTTM** is purely numeric, with no internal separators between the fields so no translation is needed. The other value that must be within the start and termination DTG range is **GFM_ORG_ORGT_M_DTG** and its template is not shown because of its complexity. It is based on figur 27.

3.3.5 Detection of Mandatory Elements

Many of the tests that have been discussed thus far have checked referential integrity, e.g., do both endpoints of a link exist? There is another requirement that says certain elements must contain a reference to another element. The next pair of tests ensure that the referring element exists. The "must-have-ote" template, shown in figur 28, find all **OBJ_TYPE** elements where the **OBJ_TYPE_ID** is not equal to an **OBJ_TYPE_ESTAB**'s **ESTABD_OBJ_TYPE_ID**. In other words, each Object Type must have an Object Type Establishment.

A similar template, using the "must-have-oiote" mode, tests the requirement that every Object Item must be linked to an Object Type with an Object Item Object Type Establishment (**OBJ_ITEM_OBJ_TYPE_ESTAB** or OIOTE). This template is not shown.

3.3.6 Type Associated With Proper Item

The "must-have-oiote" template ensures that every Object Item is associated with an Object Type. It is also important to verify that the Item and Type are compatible. For example, a **GFM_CREW_PLATFORM** must be associated with a **GFM_CREW_PLATFORM_TYPE**. The Object Type to Object Item mappings are shown in table 1. Notice that a **GOVT_ORG_TYPE** may be a leaf node in the data hierarchy only if it has a category code of `INTCIV`, `INTCMI`, or `NATCIV`.

```
<!--
  Pseudocode to explain this template.
  if (S_DTG >= T_DTG)
    report bad-dtg-range error
  else if not null(EFFCTV_DTTM)
  {
    if (not (S_DTG > EFFCTV_DTTM) and (EFFCTV_DTTM < T_DTG))
      do nothing
    else
      report date-outside-range error
  }
-->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE_ESTAB"
  mode="dtg">
  <xsl:choose>
    <xsl:when
      test="number(translate(gfm:GFM_OBJ_TYPE_ESTAB_S_DTG, '-T:Z', '')) >=
            number(translate(gfm:GFM_OBJ_TYPE_ESTAB_T_DTG, '-T:Z', ''))">
      <xsl:call-template name="bad-dtg-range">
        ...
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="gfm:EFFCTV_DTTM">
      <xsl:choose>
        <xsl:when
          test="not(number(translate(gfm:GFM_OBJ_TYPE_ESTAB_S_DTG, '-T:Z', '')) &gt;
                number(gfm:EFFCTV_DTTM)) and
                number(gfm:EFFCTV_DTTM) &lt;
                number(translate(gfm:GFM_OBJ_TYPE_ESTAB_T_DTG, '-T:Z', ''))">
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="date-outside-range">
            <xsl:with-param name="class"       select="'OBJ_TYPE_ESTAB'"/>
            <xsl:with-param name="field-name"  select="'EFFCTV_DTTM'"/>
            <xsl:with-param name="field-value" select="gfm:EFFCTV_DTTM"/>
            <xsl:with-param name="s-dtg"
                            select="gfm:GFM_OBJ_TYPE_ESTAB_S_DTG"/>
            <xsl:with-param name="t-dtg"
                            select="gfm:GFM_OBJ_TYPE_ESTAB_T_DTG"/>
            <xsl:with-param name="ewid"        select="gfm:OBJ_TYPE_ESTAB_IX"/>
            <xsl:with-param name="name"        select="''"/>
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

Figure 27. Template that find invalid effective datetime values.

```
<!-- find OBJ_TYPE that does not have an OBJ_TYPE_ESTAB -->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE"
  mode="must-have-ote">
  <xsl:choose>
    <xsl:when test="gfm:OBJ_TYPE_ID=
                  //gfm:OBJ_TYPE_ESTAB/gfm:ESTABD_OBJ_TYPE_ID">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="missing-obj">
        <xsl:with-param name="class"      select="'OBJ_TYPE'"/>
        <xsl:with-param name="miss-class" select="'OBJ_TYPE_ESTAB'"/>
        <xsl:with-param name="ewid"       select="gfm:OBJ_TYPE_ID"/>
        <xsl:with-param name="name"       select="gfm:NAME_TXT"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 28. Template that find Object Types without Establishments.

Table 1. Object Type to Object Item associations.

Object Type	**Object Item**		
	Unit	Crew Platform	Billet
Exctv Mil Org Type	X		
Unit Type	X		
Task Frmtn Type	X		
Crew Platform Type		X	
Civ Post Type			X
Mil Post Type			X
Prv Sctr Org Type	X		
Group Org Type	X		
Govt Org Type			
INTCIV	X		
INTCMI	X		
NATCIV	X		

A portion of the "item-matches-type" template is shown in figur 29. The outer conditional determines the type of the Object Item, and each inner test verifie that Object Type is an allowable type. The code for the Crew Platform is shown because it is the simplest.

```
<!-- find OBJ_ITEM that does not link to proper type of OBJ_TYPE -->
<xsl:template priority="2"
  match="//gfm:OBJ_ITEM_OBJ_TYPE_ESTAB"
  mode="item-matches-type">
  <xsl:variable name="item-ewid" select="gfm:OBJ_ITEM_ID"/>

  <xsl:choose>
    <xsl:when
      test="gfm:OBJ_ITEM_ID=//gfm:GFM_CREW_PLATFORM/gfm:GFM_CREW_PLATFORM_ID">
      <xsl:choose>
        <xsl:when
          test="gfm:ESTABD_OBJ_TYPE_ID=
              //gfm:GFM_CREW_PLATFORM_TYPE/gfm:GFM_CREW_PLATFORM_TYPE_ID">
        </xsl:when>
        <xsl:otherwise>
          <xsl:call-template name="item-type-mismatch">
            <xsl:with-param name="item-name"
                            select=
                "//gfm:OBJ_ITEM[gfm:OBJ_ITEM_ID=$item-ewid]/gfm:NAME_TXT"/>
            <xsl:with-param name="item-ewid"
                            select="$item-ewid"/>
            <xsl:with-param name="item-class"
                            select="' GFM_CREW_PLATFORM_TYPE'"/>
            <xsl:with-param name="oiote-ewid"
                            select="gfm:OBJ_ITEM_OBJ_TYPE_ESTAB_IX"/>
          </xsl:call-template>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:when>
    ...
  </xsl:choose>
</xsl:template>
```

Figure 29. Template that find Object Items with incorrect Object Types.

3.3.7 Consistent References

In the JC3 model, an **OBJ_TYPE_ESTAB** is an index to an **OBJ_TYPE**, and together they make up a compound key. Each Establishment may be thought of as a numbered variant on an Object Type. All references are required to provide both the **OBJ_TYPE_ID** and the **OBJ_TYPE_ESTAB_IX** to uniquely identify the Object Type Establishment.

GFM replaces the integer index with an EwID. The compound keys still function correctly, but redundancy has been introduced to the references as shown in figur 30. In this example, the **OBJ_ITEM_OBJ_TYPE_ESTAB** directly references the **OBJ_TYPE** and indirectly via the **OBJ_TYPE_ESTAB**. The template with the mode “objtype-matches-estab” ensures that the **OBJ_TYPE** in a reference has the same value as the **OBJ_TYPE** in the referenced **OBJ_TYPE_ESTAB**. One of the templates for mode “objtype-matches-estab” is shown in figur 31.

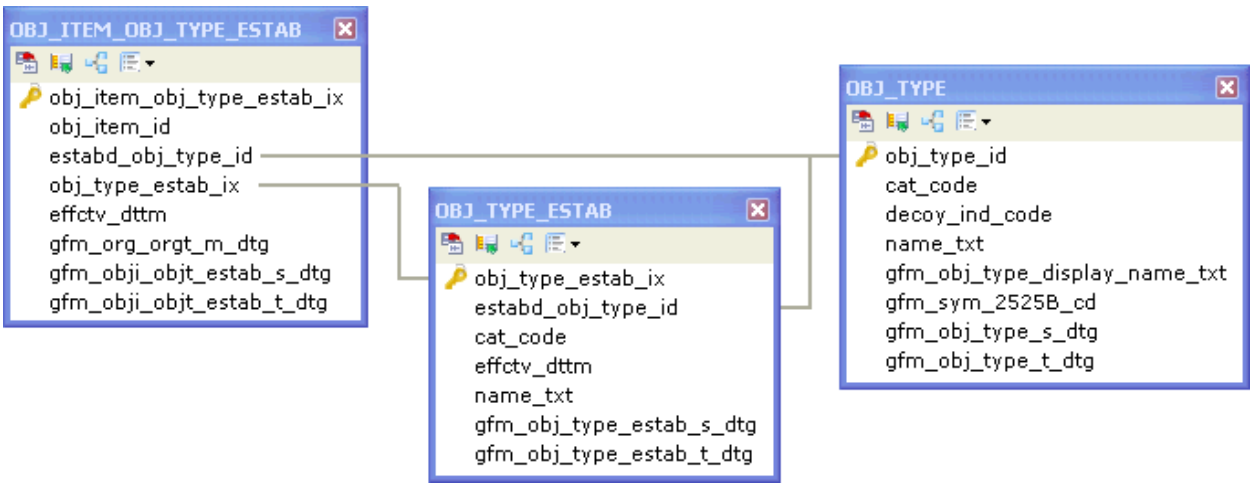

Figure 30. Redundant reference.

```
<!-- OBJ_TYPE/ESTAB references must match the OTE's OBJ_TYPE -->
<!-- check OIOTEs because they have dual references -->
<xsl:template priority="2"
  match="//gfm:OBJ_ITEM_OBJ_TYPE_ESTAB"
  mode="objtype-matches-estab">
  <xsl:variable name="estab-ewid"
  select="gfm:OBJ_TYPE_ESTAB_IX"/>
  <xsl:choose>
    <xsl:when
      test="gfm:ESTABD_OBJ_TYPE_ID=//gfm:OBJ_TYPE_ESTAB
      [gfm:OBJ_TYPE_ESTAB_IX=$estab-ewid]/gfm:ESTABD_OBJ_TYPE_ID">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="objtype-estab-mismatch">
        <xsl:with-param name="link-class"
                        select="'OBJ_ITEM_OBJ_TYPE_ESTAB '"/>
        <xsl:with-param name="link-ewid"
                        select="gfm:OBJ_ITEM_OBJ_TYPE_ESTAB_IX"/>
        <xsl:with-param name="estab-ewid"
                        select="$estab-ewid"/>
        <xsl:with-param name="objtype-ewid"
                        select= "gfm:ESTABD_OBJ_TYPE_ID"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 31. Template that find Object Items with incorrect Object Types.

3.3.8 Single Root Node

Each fil is expected to contain an Organisation tree root node and, optionally, an OrgType tree root node. While it is possible for an XML dump of a GFM database to contain multiple roots

with different date/time intervals, this is unlikely. The template with the mode “roots” for Organisation trees is shown in figur 32.

```
<!-- count Org roots -->
<xsl:template priority="2"
  match="//gfm:OBJ_ITEM_ASSOC_TBL"
  mode="roots">
  <xsl:variable name="root-assocs"
    select="gfm:OBJ_ITEM_ASSOC[gfm:SUBJ_OBJ_ITEM_ID=gfm:OBJ_OBJ_ITEM_ID]"/>
  <xsl:if test="count($root-assocs) &gt; 1">
    <xsl:call-template name="too-many-roots">
      <xsl:with-param name="class-name" select="'Org'"/>
      <xsl:with-param name="link-list"  select="$root-assocs"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Figure 32. Template that find multiple Organisation tree roots.

This template stores all of the Assocs where the parent and child node are the same, completely ignoring the date/time interval. If the number of Assocs is greater than one, the list is passed to the output template for printing.

3.3.9 Putting the Parts Together

A template near the top of the XSLT fil is the main driver.[18] It writes the HTML wrapper for the output fil and applies all of the match templates with their modes. It is shown in figur 33 with most of the tests removed.

The contents of the ‘title’ and ‘h1’ tags are identical. They display text describing the output and show the file s ‘TITLE’ attribute (assuming that one exists). The ‘choose’ conditional applies the “tree” templates only if the fil contains object-oriented data elements. The second mode (“link”) shows the form used by the rest of the template sets.

The code to determine if the fil contains object-oriented data elements counts the number of “OO_TBL” elements found, as shown in figur 34. The variable **hierCount** is assigned a value from 0–3; a zero means the fil is in relational form.

Partitioning the templates by assigning them to different modes helps to prevent unwanted overlaps in the templates. It also allows the user to remove the ‘apply-templates’ element of any unwanted test that may be deemed too time-consuming.

[18] The order of the templates in the fil is irrelevant and was chosen for readability.

```
<!-- root writes general HTML wrapper and processes all data -->
<xsl:template match="/">
  <html>
    <head>
      <title>Results of Validation (using XSLT)
        <xsl:value-of select="//gfm:GFMIEDM34/@TITLE"/>
      </title>
    </head>
    <body>
      <h1>Results of Validation (using XSLT)
        <xsl:value-of select="//gfm:GFMIEDM34/@TITLE"/>
      </h1>
      <h2>Improper category/child element</h2>
    <!-- perform test only for hierarchical data -->
      <xsl:choose>
        <xsl:when test="$hierCount > 0">
          <ol>
            <xsl:apply-templates mode="tree"/>
          </ol>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>Tests may not be performed on relational data</xsl:text>
        </xsl:otherwise>
      </xsl:choose>
      <h2>Links with missing endpoint(s)</h2>
      <ol>
        <xsl:apply-templates mode="link"/>
      </ol>
      ...
    </body>
  </html>
</xsl:template>
```

Figure 33. Main (root) template.

```
<!-- count the number of hierarchical tables (0-3) -->
<xsl:variable name="hierCount"
  select="count(//gfm:GFMIEDM34/gfm:OBJ_TYPE_OO_TBL) +
          count(//gfm:GFMIEDM34/gfm:OBJ_ITEM_OO_TBL) +
          count(//gfm:GFMIEDM34/gfm:ADDR_OO_TBL)"/>
```

Figure 34. Counting the number of object-oriented tables.

3.4 Validation of Business Rules

3.4.1 Introduction

The file GFMIEDM341businessRules.xsl checks a GFM XML data file for conformance with rules formulated by the GFM user community. While a particular enumerated value may be in the JC3 schema, it may not be permitted by the GFM rules. Other templates test for compliance with

higher level constraints, such as the fact that a Billet may not have a child in the Organisation tree. Because of the complexity of the tests, most of the templates contain procedural code. The fil implements seven modes, the firs of which is named “link”.

3.4.2 Link Type Validation

The GFM XSD ensures that the values assigned to **CAT_CODE** and **SUBCAT_CODE** elements are allowable enumerated values. The “link” templates in this fil further restrict the values to pairs that conform to the business rules. Because of the complexity of the code, “good” templates were defined One of these is shown in figur 35, with the equivalent SQL code in figur 36.

```
<!-- ignore nodes where cat_code/subcat_code pair is allowed -->
<!-- ORG_TYPE to ORG_TYPE links -->
<!-- we only need to check the child's object type -->
<xsl:template priority="3"
  match="//gfm:OBJ_TYPE_ESTAB_OBJT_DET
          [gfm:GFM_OTEOD_CAT_CODE='HSADMI' and
           gfm:GFM_OTEOD_GFM_SUBCAT_CODE='DEFALT' and
           gfm:DET_OBJ_TYPE_ID=//gfm:ORG_TYPE/gfm:ORG_TYPE_ID]"
  mode="link">
</xsl:template>
```

Figure 35. Template that matches OTEOD links with valid code pairs.

```
SELECT COUNT(*)
  FROM OBJ_TYPE_ESTAB_OBJT_DET oteod,
       ORG_TYPE ot
  WHERE oteod.GFM_OTEOD_CAT_CODE='HSADMI' AND
        oteod.GFM_OTEOD_GFM_SUBCAT_CODE='DEFALT' AND
        oteod.DET_OBJ_TYPE_ID=ot.ORG_TYPE_ID;
```

Figure 36. SQL query that find OTEOD links with valid code pairs.

This template enforces the rule “all OTEOD links between two Org Types must use the codes “HSADMI/DEFALT.” Only the type of the child is checked under the assumption that Org Types always have an Org Type parent (see section 3.4.6). Other templates test the remaining allowable code pairs.

Notice that the priority of the template has been raised to 3, and the template has no contents. All OTEOD links that have acceptable codes will be matched and processed, producing no output. The remaining OTEOD links will match the template in figur 37.

The latter template is complicated because it must determine which codes are invalid based on the override policy. There are 8 templates that match good OTEODs and Assocs, and 2 templates that catch the remaining OTEODs and Assocs and produce error messages. As in the other XSLT file a template with a priority of -1 catches all text elements that are not OTEODs or Assocs and ignores them.

```
<!-- locate nodes where cat_code/subcat_code pair is not allowed -->
<!-- all good nodes have already been matched -->
<!-- templates are complicated because of override rules -->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE_ESTAB_OBJT_DET"
  mode="link">
  <xsl:choose>
    <xsl:when test="gfm:GFM_OTEOD_GFM_CAT_CODE='NOS' and
                    gfm:GFM_OTEOD_GFM_SUBCAT_CODE='NOS'">
      <xsl:call-template name="illegal-link">
        <xsl:with-param name="class"
                        select="'OBJ_TYPE_ESTAB_OBJT_DET'"/>
        <xsl:with-param name="catcode"
                        select="gfm:GFM_OTEOD_CAT_CODE"/>
        <xsl:with-param name="subcatcode"
                        select="gfm:GFM_OTEOD_SUBCAT_CODE"/>
        <xsl:with-param name="ewid"
                        select="gfm:OBJ_TYPE_ESTAB_OBJT_DET_IX"/>
        <xsl:with-param name="label" select="gfm:GFM_OTEOD_LABEL_TXT"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="gfm:GFM_OTEOD_GFM_CAT_CODE='NOS' and
                    gfm:GFM_OTEOD_GFM_SUBCAT_CODE!='NOS'">
      <xsl:call-template name="illegal-link">
        ...
      </xsl:call-template>
    </xsl:when>
    <xsl:when test="gfm:GFM_OTEOD_GFM_CAT_CODE!='NOS' and
                    gfm:GFM_OTEOD_GFM_SUBCAT_CODE='NOS'">
      <xsl:call-template name="illegal-link">
        ...
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="illegal-link">
        ...
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 37. Template that matches OTEOD links with invalid code pairs.

3.4.3 Person Type Category Code Validation

The templates that validate Person Type codes are simple. They test the requirement that "military Person Types must have a subcategory of NOS, while civilian Person Types must have a subcategory of GOVEMP or NONGVE." There are two "bad" templates like the one shown in figur 38. The explanation for how this code works is described with figur 24.

```
<!-- Person Types must have proper cat_codes -->
<xsl:template priority="2"
  match="//gfm:PERS_TYPE[gfm:CAT_CODE='MILTRY'][gfm:SUBCAT_CODE!='NOS']"
  mode="person">
  <xsl:call-template name="mil-civ">
    <xsl:with-param name="type"       select="'Military'"/>
    <xsl:with-param name="correct"    select="'MILTRY/NOS'"/>
    <xsl:with-param name="catcode"    select="gfm:CAT_CODE"/>
    <xsl:with-param name="subcatcode" select="gfm:SUBCAT_CODE"/>
    <xsl:with-param name="ewid"       select="gfm:PERS_TYPE_ID"/>
    <xsl:with-param name="skill-code" select="gfm:GFM_PERS_TYPE_SKILL_CD"/>
  </xsl:call-template>
</xsl:template>
```

Figure 38. Template that checks Person Type categories.

3.4.4 Organisation Validation

Every Organisation (ORG) must appear in the Organisation tree. Any Organisation that does not have a parent (i.e., is an orphan) is assumed to be a mistake and should be reported. The template in figur 39 find and discards all ORGs that are children of Assoc links. The remaining ORGs match the template in figur 40 and an error message is generated.

```
<!-- ignore Orgs which are children -->
<xsl:template priority="3"
  match="//gfm:ORG[gfm:ORG_ID=//gfm:OBJ_ITEM_ASSOC/gfm:OBJ_OBJ_ITEM_ID]"
  mode="assoc">
</xsl:template>
```

Figure 39. Template that find ORGs that are children.

```
<!-- every Org must be a child -->
<xsl:template priority="2"
  match="//gfm:ORG"
  mode="assoc">
  <xsl:call-template name="never-child">
    <xsl:with-param name="ewid" select="gfm:ORG_ID"/>
    <xsl:with-param name="name" select="gfm:GFM_ORG_SHORT_NAME_TXT"/>
  </xsl:call-template>
</xsl:template>
```

Figure 40. Template that reports ORGs that are not children.

3.4.5 Billet Validation

This business rule is almost the opposite of the previous one. A Billet may never be a parent node in the Organisation tree. Figure 41 find all ORGs that break this rule.

```
<!-- find billets which are parents -->
<xsl:template priority="2"
  match="//gfm:ORG[gfm:ORG_ID=//gfm:OBJ_ITEM_ASSOC/gfm:SUBJ_OBJ_ITEM_ID]
                  [gfm:ORG_ID=//gfm:GFM_BILLET/gfm:GFM_BILLET_ID]"
  mode="billet">
  <xsl:call-template name="billet-parent">
    <xsl:with-param name="ewid" select="gfm:ORG_ID"/>
    <xsl:with-param name="name" select="gfm:GFM_ORG_SHORT_NAME_TXT"/>
  </xsl:call-template>
</xsl:template>
```

Figure 41. Template that find Billets that have children.

The firs part of the XPath expression matches all ORGs that are in the parent (**OBJ_OBJ_ITEM_ID**) element of an **OBJ_ITEM_ASSOC**. The second half requires that the same ORG is a **GFM_BILLET**. An ORG that meets both of these criteria is therefore a Billet that is a parent of another Object Item.

3.4.6 Link Endpoint Types

The force structure tree for organisation types consists of Org Type parents and children. The links are contained in the **OBJ_TYPE_ESTAB_OBJT_DET_TBL** element along with other kinds of links. Materiel, Person, and Facility Types may be aligned with an Org Type via an OTEOD link, while Materiel and Person Types may be clustered using OTEODs. The allowable endpoint type combinations are shown in table 2.

Table 2. Allowable OTEOD combinations.

		Child			
		Org Type	Mat Type	Pers Type	Fac Type
Parent	Org Type	X	X	X	X
	Mat Type		X		
	Pers Type			X	

Because an Org Type may have any type of child, explicit testing is not needed; other link tests will catch any errors. However, links with Mat Type and Person Type parents must be checked. In addition, the establishments of the latter two parent Object Types are required to have a category code of “PCG” (Parts Catalogue). The template for locating and testing Mat Type parents is shown in figur 42. The Person Type template is identical with element names changed as appropriate.[19]

[19] Additional templates should be written to thoroughly test Person Type clusters.

```
<!-- find all OTEODs with a MatType parent, then check the OTEOD's child -->
<xsl:template priority="2"
  match="//gfm:OBJ_TYPE_ESTAB_OBJT_DET[gfm:ESTABD_OBJ_TYPE_ID=
         //gfm:MAT_TYPE/gfm:MAT_TYPE_ID]"
  mode="oteod">
  <xsl:choose>
    <xsl:when test="gfm:DET_OBJ_TYPE_ID=//gfm:MAT_TYPE/gfm:MAT_TYPE_ID">
      <xsl:call-template name="check-for-PCG">
        <xsl:with-param name="ote-ewid" select="gfm:OBJ_TYPE_ESTAB_IX"/>
      </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="same-parent-child">
        <xsl:with-param name="link-ewid"
                        select="gfm:OBJ_TYPE_ESTAB_OBJT_DET_IX"/>
        <xsl:with-param name="label" select="gfm:GFM_OTEOD_LABEL_TXT"/>
        <xsl:with-param name="class" select="'MAT_TYPE'"/>
        <xsl:with-param name="parent-ewid" select="gfm:ESTABD_OBJ_TYPE_ID"/>
        <xsl:with-param name="child-ewid"  select="gfm:DET_OBJ_TYPE_ID"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 42. Template that checks child of Mat Type parent.

The XPath expression matches all OTEODs where the parent (**ESTAB_OBJ_TYPE_ID**) is a Mat Type. If the OTEOD's child (**DET_OBJ_TYPE_ID**) is also a Mat Type, then the named template "check-for-PCG" is called, passing the EwID of the parent Object Type Establishment (**OBJ_TYPE_ESTAB_IX**) as a parameter. Any other type of child produces an error message.

The named template is procedural code and is shown in figur 43. It begins by findin the Object Type Establishment that has the desired EwID and storing the value of its category code in the variable **catcode**. If the variable does not have the value PCG, then an error is generated.

```
<xsl:template name="check-for-PCG">
  <xsl:param name="ote-ewid"/>
  <xsl:variable name="catcode"
    select="//gfm:OBJ_TYPE_ESTAB[gfm:OBJ_TYPE_ESTAB_IX=
           $ote-ewid]/gfm:CAT_CODE"/>
  <xsl:if test="$catcode!='PCG'">
    <xsl:call-template name="parent-ote-not-pcg">
      <xsl:with-param name="ote-ewid" select="$ote-ewid"/>
      <xsl:with-param name="catcode"  select="$catcode"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Figure 43. Named template that checks the category code of an Object Type Establishment.

3.4.7 Required Associations

The next three tests have similar requirements, and therefore the templates are identical except for changes in element names. The firs template, whose mode is "crewtype", determines if every **CREW_PLATFORM_TYPE** (CPT) has a Mat Type aligned with it.[20] The template collects all of the CPTs, then locates at least one link where each CPT is the parent and a Mat Type is the child. The template is shown in figur 44.

```
<!-- find all CPTs, then look for an OTEOD with a MatType child -->
<xsl:template priority="2"
  match="//gfm:GFM_CREW_PLATFORM_TYPE"
  mode="crewtype">
  <xsl:variable name="cpt-ewid" select="gfm:GFM_CREW_PLATFORM_TYPE_ID"/>
  <xsl:choose>
    <xsl:when
      test="//gfm:OBJ_TYPE_ESTAB_OBJT_DET
            [gfm:ESTABD_OBJ_TYPE_ID=$cpt-ewid]
            [gfm:DET_OBJ_TYPE_ID=//gfm:MAT_TYPE/gfm:MAT_TYPE_ID]">
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template name="sometype-does-not-have-sometype">
        <xsl:with-param name="ewid" select="$cpt-ewid"/>
        <xsl:with-param name="type" select="'MatType'"/>
      </xsl:call-template>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Figure 44. Template that reports Crew Platform Types that do not have a Mat Type.

The related pair of templates, using the mode "posttype", perform the same logic with **CIV_POST_TYPE** and **MIL_POST_TYPE** objects and an assigned mandatory **PERS_TYPE** object. They are not shown. By passing part of the diagnostic message as a parameter, the same named output template may be used by all three templates.

3.4.8 Aligning Person Types

The required Person Type objects may be aligned with each **MIL_POST_TYPE** and **CIV_POST_TYPE** object in one of two different ways:

1. Each PersType may be aligned directly to a PostType with an OTEOD, or
2. A set of PersTypes may be *clustered* into a PersType tree (PTT).

The second has the advantage of allowing commonly used sets of qualifications such as a Radio Operator, to be define once and then linked to the appropriate MilPostTypes with a single OTEOD.

[20]The Post Types subordinate to this object are carried by the Mat Type, which should be a vehicle of some type. This latter test is not performed by the code.

There are specifi requirements for a PTT. The firs is the root PersType of the PTT must reference a Person Type Skill Attribute (PTSA) with a type code of ROS, which stands for "Root Occupational Specialty". The template, which has a mode of "ros", is shown in figur 45.

```
<!-- find all PersTypes that are parents and make sure PTSA is ROS. -->
<xsl:template priority="2"
  match="//gfm:PERS_TYPE[gfm:PERS_TYPE_ID=
         //gfm:OBJ_TYPE_ESTAB_OBJT_DET/gfm:ESTABD_OBJ_TYPE_ID]"
  mode="ros">
  <xsl:variable name="ptsa-ewid" select="gfm:GFM_PERS_TYPE_SKILL_ATTS"/>
  <xsl:variable name="catcode"
    select="//gfm:GFM_PERS_TYPE_SKILL_ATTR[gfm:GFM_PERST_SKILL_ATTR_ID=
            $ptsa-ewid]/gfm:GFM_PERST_SKILL_ATTR_TYPE_CD"/>
  <xsl:if test="$catcode!='ROS'">
    <xsl:variable name="pt-ewid" select="gfm:PERS_TYPE_ID"/>
    <xsl:call-template name="not-ros">
      <xsl:with-param name="catcode"    select="$catcode"/>
      <xsl:with-param name="ewid"       select="gfm:PERS_TYPE_ID"/>
      <xsl:with-param name="name"
        select="//gfm:OBJ_TYPE[gfm:OBJ_TYPE_ID=$pt-ewid]/gfm:NAME_TXT"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Figure 45. Template that reports Person Type roots that do not have a PTSA of ROS.

Variables are used to circumvent restrictions imposed by XPath.[21] The template matches all PersTypes that appear as the parent of an OTEOD. The PTSA of the PersType is stored in a variable, then the variable is used in an expression to fetch the type code of that PTSA. This value is examined to see if it is not ROS, in which case an error message is generated.

The second PTT requirement is the number of PersTypes that must be included in a PersType tree. The template shown in figur 46, with the mode "cluster", counts the number of PTT nodes and reports errors.

The template begins by findin all PTSAs that have a type code of ROS. The code loops through each PersType that references that PTSA, counting the number of child PersTypes and storing the value in a variable. Military PTTs must have 5 children, while civilian trees are required to have 3 child PersTypes. If the number is incorrect, an appropriate message is produced.

This template overlaps other tests. It assumes that the children are all PersTypes (see section 3.4.6) and that PersType roots are always ROS. If those conditions are not met, this test may give incorrect results. Either way, mistakes will be detected and reported.

The test with mode="count-perstypes" in figur 47 check the number of Person Types that are directly aligned with a MilPostType.[22] This is the most complex of the GFM tests, using multiple variables define with expressions.

[21] This is equivalent to a 3-way join in SQL.

[22] A similar test checks CivPostTypes.

```
<!-- find all PTSAs with TYPE_CD='ROS'.  -->
<!-- for each PersType that uses that PTSA, count the children.  -->
<xsl:template priority="2"
  match="//gfm:GFM_PERS_TYPE_SKILL_ATTR[
         gfm:GFM_PERST_SKILL_ATTR_TYPE_CD='ROS']"
  mode="cluster">
  <xsl:variable name="ptsa-ewid" select="gfm:GFM_PERST_SKILL_ATTR_ID"/>
  <xsl:for-each
    select="//gfm:PERS_TYPE[gfm:GFM_PERS_TYPE_SKILL_ATTS=$ptsa-ewid]">
    <xsl:variable name="ros-ewid" select="gfm:PERS_TYPE_ID"/>
    <xsl:variable name="type"     select="gfm:CAT_CODE"/>
    <xsl:variable name="num-children"
      select="count(//gfm:OBJ_TYPE_ESTAB_OBJT_DET[
              gfm:ESTABD_OBJ_TYPE_ID=$ros-ewid])"/>
    <xsl:if test="(($type='MILTRY')  and ($num-children!=5)) or
                  (($type!='MILTRY') and ($num-children!=3))">
      <xsl:call-template name="bad-perstype-tree">
        <xsl:with-param name="type"     select="gfm:CAT_CODE"/>
        <xsl:with-param name="ros-ewid" select="$ros-ewid"/>
        <xsl:with-param name="name"
          select="//gfm:OBJ_TYPE[gfm:OBJ_TYPE_ID=$ros-ewid]/gfm:NAME_TXT"/>
        <xsl:with-param name="num-children" select="$num-children"/>
      </xsl:call-template>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Figure 46. Template that reports Person Type trees that have an incorrect number of nodes.

The template begins by matching all MilPostType elements and storing the ID of each one. The list of OTEOD links where the parent is the current MilPostType is stored in a second variable. The next variable contains a temporary tree containing 'child' elements; the name chosen is irrelevant. A loop iterates through all of the PersTypes that are children of the OTEOD links, and a 'child' element is created for each PersType. The contents of the child are empty or contain a PTSA. The last variable looks for a child element that is not empty. If such an element is found, it means that the PostType links to a PTT, and the value 5 is stored. Otherwise, the number of links is counted and stored.

The fina step checks the number of children that were found. If fewer than 5 PersType children were found, an error message is generated. A PostType that has no child PersTypes is ignored because that case is tested with the mode "posttype" as described in section 3.4.7.

3.4.9 General Information

A simple test verifie that an XML data fil contains at least one Crew Platform and at least one Billet.[23] The template in figur 48 counts all GFM_CREW_PLATFORM elements in the XML file If no Crew Platforms are found, a special message is produced, otherwise, the actual number

[23] These are general suggestions and may not always be applicable.

found is displayed. The second half of the test, which counts Billets, is identical except for the fiel name and generated text.

```
<!-- MilPostType must have 5 simple PersTypes -->
<!-- for each MPT, find all OTEODs where the child is a PersType -->
<xsl:template priority="2"
  match="//gfm:MIL_POST_TYPE"
  mode="count-perstypes">
  <xsl:variable name="mpt-ewid" select="gfm:MIL_POST_TYPE_ID"/>
  <xsl:variable name="oteods"
    select="//gfm:OBJ_TYPE_ESTAB_OBJT_DET[gfm:ESTABD_OBJ_TYPE_ID=
      $mpt-ewid][gfm:DET_OBJ_TYPE_ID=//gfm:PERS_TYPE/gfm:PERS_TYPE_ID]"/>
<!-- is any child an ROS? -->
  <xsl:variable name="found-ros">
    <xsl:for-each
      select="//gfm:PERS_TYPE[gfm:PERS_TYPE_ID=$oteods/gfm:DET_OBJ_TYPE_ID]">
      <xsl:variable name="ptsa-ewid" select="gfm:GFM_PERS_TYPE_SKILL_ATTS"/>
      <child>
        <xsl:value-of
          select="//gfm:GFM_PERS_TYPE_SKILL_ATTR[gfm:GFM_PERST_SKILL_ATTR_ID=
            $ptsa-ewid][gfm:GFM_PERST_SKILL_ATTR_TYPE_CD='ROS']"/>
      </child>
    </xsl:for-each>
  </xsl:variable>
<!-- compute or store number of children -->
  <xsl:variable name="num-children">
    <xsl:choose>
      <xsl:when test="$found-ros[child!='']">
        5
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="count($oteods)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
<!-- required number of PersType children found? -->
  <xsl:if test="$num-children > 0 and $num-children &lt; 5">
    <xsl:call-template name="wrong-number-perstypes">
      <xsl:with-param name="title" select="'MIL_POST_TYPE '"/>
      <xsl:with-param name="type"  select="'MILTRY'"/>
      <xsl:with-param name="ros-ewid" select="$mpt-ewid"/>
      <xsl:with-param name="name"
        select="//gfm:OBJ_TYPE[gfm:OBJ_TYPE_ID=$mpt-ewid]/gfm:NAME_TXT"/>
      <xsl:with-param name="desc" select="'alignments'"/>
      <xsl:with-param name="num-children" select="$num-children"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Figure 47. Template that reports MilPostTypes with too few PersType children.

This test has inspired the creation of yet another XSLT fil for GFM. The fil GFMIEDM341.xsl counts all of the major elements in an XML fil and displays them at the top of the XHTML fil

of the pretty-printed data, while the IChart application produces a more detailed breakout in its log file A fairly simple XSLT fil has been written (but not released at this time) that produces a summary of the data in an XML fil to help validate the data.

```
<!-- Count number of CrewPlatforms (and Billets) -->
  <xsl:template priority="2" match="*"
    mode="count">
    <xsl:variable name="num-crews" select="count(//gfm:GFM_CREW_PLATFORM)"/>
    <li>
      <xsl:choose>
        <xsl:when test="$num-crews=0">
          <xsl:text>There should be at least 1 Crew Platform</xsl:text>
        </xsl:when>
        <xsl:otherwise>
          <xsl:text>Found </xsl:text>
          <xsl:value-of select="$num-crews"/>
          <xsl:text> Crew Platform</xsl:text>
          <xsl:if test="$num-crews > 1">
            <xsl:text>s</xsl:text>
          </xsl:if>
        </xsl:otherwise>
      </xsl:choose>
    </li>
</xsl:template>
```

Figure 48. Template that counts Crew Platform (and Billet) elements.

4. Performing Validation Testing

4.1 Schema Validation

Every XML data fil should have a reference to the physical fil that contains the XSD. The root element used by GFM XML file is shown in figur 49.

```
<GFMIEDM34 TITLE="optional description of data"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ism="urn:us:gov:ic:ism:v2"
xmlns="http://gfm.arl.army.mil/GFMIEDM34"
xsi:schemaLocation="http://gfm.arl.army.mil/GFMIEDM34
  GFMIEDM341.xsd">
  ... data elements ...
</GFMIEDM34>
```

Figure 49. Typical GFM root element.

The actual file name is highlighted. The five files that make up the GFM XSD plus the IC-ISM XSD file (see figure 3) must all be in the same folder. If the XML data file is in a different folder, then the path to the XSD file must be added to the highlighted file name in figure 49.

The validation may be performed with the Xerces-J (*16*) Java Archive (JAR) files. Figure 50 shows the command line (in two parts for readability) to validate a sample file that contains one error.

```
java -classpath xerces.jar;xercesSamples.jar
  sax.SAXCount -s -v -f -an somefile.xml

[Error] somefile.xml:1183:17: Key with value [ID Value: 7205759403792784] not
        found for identity constraint of element "OBJ_ITEM".
somefile.xml: 1112 ms (1155 elems, 182 attrs, 7205 spaces, 11102 chars)
```

Figure 50. Xerces XML validator detecting one error.

The public Java application SAXCount was written to count objects in an XML file. One available option is to also validate the data if an XSD is available. In this example[24], an error was found on line 1183, column 17. While the error message may appear to be cryptic, an examination of the OBJ_ITEM shows that the value `7205759403792784` is missing a trailing zero. The error report is followed by a line of summary information.

Other tools include XMLSpy® (*17*), a commercial product that graphically edits and validates XML files. The nice feature of XMLSpy is that it actually highlights the element that contains an error; the bad thing is it stops at the first error that it finds.[25] Saxon (*18*) is an XSLT transformation engine that is run from the command line, and the commercial version also performs XSD validation.[26] ARL's IChart (*19*) application always tests GFM XML data files for well-formedness, and it will optionally test files for validity using Xerces-J.

4.2 XSLT Validation

Unlike XSD, XSLT was not designed to perform validation, so the process is slightly more involved. Either Saxon or XMLSpy may be used as the transformation engine. The command line for Saxon is shown in figure 51.

```
java -jar saxon9.jar somefile.xml GFMIEDM341validate.xsl >err.html

Warning: at xsl:stylesheet on line 107 of file:GFMIEDM341validate.xsl:
  Running an XSLT 1.0 stylesheet with an XSLT 2.0 processor
```

Figure 51. Saxon transformation engine testing XML data.

[24] I used Xerces-J version 1.4.4. With appropriate changes to figure 50, Xerces-J version 2.9.1 gives identical results.
[25] Xerces attempts to find all of the errors at one time.
[26] I have not used the commercial version.

The warning message is a reminder that there is a mismatch between the transformation engine and the XSLT file Saxon supports the new XSLT 2.0 and XPath 2.0 specifications while the GFM XSLT fil was intentionally limited to 1.0 features to ensure portability.

The output of the transformation has been stored in the fil err.html (or whatever fil name the user supplied). A normal text fil could have been generated, but this leverages off of the formatting performed by XHTML. The next step is to load err.html into a Web browser.

Several GFM XML data file have been created to test the XSLT validation files They are available to the user community for anyone who wants to confir that their own validation tool is working correctly. The file GFMIEDM341invalidTestsSet1.xml and GFMIEDM341invalidTestsSet2.xml contain hierarchical data to test GFMIEDM341validate.xsl, while GFMIEDM341invalidSampleRelat.xml is similar data in relational form. All three file contain errors that are described in section 3.3; however, category code errors from section 3.3.2 may be detected only in the hierarchical sample XML file

To ensure that every template reports an error, many intentional mistakes were made in the sample files The correct data was tested firs to avoid false positives, then the errors were introduced. There are 34 category code errors (only in the firs file and 57 other errors (in both files) An abridged snapshot of the Web browser output is shown in figur 52. The last line of each item contains the EwID of the offending object and identifying text (where it is available).

It is not possible for XSLT to report when no errors are found in a GFM XML data file If no output is produced, then the assumption is that there are no errors. The conditional in the top half of figur 33 explicitly states when the "tree" tests may not be performed because the GFM XML data fil contains only relational data. Figure 53 shows examples of both of these conditions using a version of the GFMIEDM341invalidSampleRelat.xml fil that contains only two errors.

XMLSpy, because of its graphical interface, is easier to use. Load a GFM XML data file assign an XSLT fil to it, and command XMLSpy to perform the transformation. The formatted XHTML output will be displayed in another tab panel.

A third method, which is not recommended but is included for completeness, is to let the Web browser's XSLT transformation engine do all of the work. The top of every GFM XML data fil should contain the line

```
<?xml-stylesheet type="text/xsl" href="GFMIEDM341.xsl"?>
```

which causes the browser to transform the data into a more readable form. Replace the highlighted fil name with the desired validation XSLT fil name, then load the data fil into the browser. Eventually you will see the error text. This is not practical for large data files but it is a handy trick if you do not have Saxon or XMLSpy at hand.

The XSLT fil that tests GFM business rules is under development as new datasets inspire new rules. The business rule test fil of GFM XML data is GFMIEDM341invalidBusiness.xml. Output produced from the current (as of this writing) version of the GFMIEDM341businessRules.xsl fil and the test XML fil is shown in figur 54. The link labels and Person Type names, e.g., "bad CAT," were define in the data to state the error that should be reported and are not generated by the XSLT script.

Results of Validation (using XSLT) Sample Hierarchical Data

Improper category/child element

1. OBJ_TYPE's CAT_CODE (MA) must match child ORG_TYPE
   72057594037927960 = Some Civilian

Links with missing endpoint(s)

1. OBJ_TYPE_ESTAB_OBJT_DET's ESTABD_OBJ_TYPE_ID (2057594037927950) does not exist
   18446744073709551013

Improper override of GFM vs JC3 fields

1. CIV_POST_TYPE's GFM_CAT_CODE (VOLUNT) does not properly override CAT_CODE (MAYOR)
   72057594037927860

Objects with bad DTG ranges or bad starting dates

1. Bad DTG range in GFM_PERS_TYPE_SKILL_ATTR
   1990-01-01T00:00:00Z is not before 1005-01-01T00:00:00Z
   18446744073709551005 = skill attr name
2. OBJ_TYPE_ESTAB's EFFCTV_DTTM
   19500101000000.000 is not within DTG range of
   1990-01-01T00:00:00Z to 2010-01-01T00:00:00Z
   18446744073709551027 =

Missing Obj Type Estabs

1. OBJ_TYPE does have an OBJ_TYPE_ESTAB
   72057594037927967 = Supply Department

Missing Obj Item/Obj Type Estabs

1. OBJ_ITEM does have an OBJ_ITEM_OBJ_TYPE_ESTAB
   72057594037927944 = Platform

Obj Items linked to incorrect Obj Types

1. Item 72337338142818342 = small crew
   does not link to a GFM_CREW_PLATFORM_TYPE
   in OBJ_ITEM_OBJ_TYPE_ESTAB 72337338142818343

References to Obj Type Estabs do not match Obj Types

1. OBJ_TYPE_ESTAB_OBJT_DET 72337338142818310
   references ObjTypeEstab 72337338142818306
   and ObjType 72337338142818311
   but ObjTypeEstab 72337338142818306
   references ObjType 72337338142818305

Multiple Roots

1. found multiple Org tree roots
   Assoc = 72337338142818348 Org = sample excmil
   Assoc = 72337338142818370 Org = international civilian

Figure 52. Web browser output of XSLT validation.

Results of Validation (using XSLT) Sample Relational Data

Improper category/child element
Tests may not be performed on relational data
Links with missing endpoint(s)
Improper override of GFM vs JC3 fields
Objects with bad DTG ranges or bad starting dates
Missing Obj Type Estabs
1. OBJ_TYPE does have an OBJ_TYPE_ESTAB
   72057594037927967 = Supply Department

Missing Obj Item/Obj Type Estabs
1. OBJ_ITEM does have an OBJ_ITEM_OBJ_TYPE_ESTAB
   72057594037927944 = Platform

Obj Items linked to incorrect Obj Types
References to Obj Type Estabs do not match Obj Types
Multiple Roots

Figure 53. Web browser output of validation of relational data.

Results of Validation (using XSLT)

Improper link types

1. OBJ_TYPE_ESTAB_OBJT_DET has illegal link type combination
   ADMINS/DEFALT
   72060759428875092 = bad CAT
2. OBJ_TYPE_ESTAB_OBJT_DET has illegal link type combination
   HSADMI/TACCOM
   72060759428875093 = bad SUBCAT

Improper Person Type codes

1. Military PERS_TYPE must be MILTRY/NOS
   Found MILTRY/PILOT
   72059647032295510 = bad mil codes
2. Civilian PERS_TYPE must be CIV/GOVEMP or CIV/NONGVE
   Found CIV/JRNLST
   72060755133882801 = bad civ codes
3. PERS_TYPE root must be ROS, found SKLLVL
   72060793789194799 = EXECUTIVE OFFICER

Orgs which are not in tree

1. ORG is never in Org Tree
   72060755133858478 = OSD-orphan

Billets which have children

1. Billet is a parent, which is not allowed
   72060755133860793 = parent org

OTEODs which have improper parent/child type

1. OBJ_TYPE_ESTAB_OBJT_DET does not have matching parent/child types
   72060759428875121 = MT parent, not MT child
   MAT_TYPE RIFLE 5.56 MM: M16A2 (72057594037927970) may not be parent of
   Head of State (72060759428875115)
2. OBJ_TYPE_ESTAB_OBJT_DET does not have matching parent/child types
   72059647032302985 = PT parent, not PT child
   PERS_TYPE JAG, O-5 (72059647032297826) may not be parent of
   WORLD (72060755133857960)
3. Parent OTE should have CAT_CODE = PCG
   27A00 O-6 (72059647032297830) has CAT_CODE = CES

CrewPlatformTypes which do not have an aligned MatType

1. 72337338142818314 = small crew
   does not have a MatType

PostTypes which do not have an aligned PersType

1. 72060755133858003 = Mayor
   does not have a PersType

PersType Tree Counts

1. MILTRY PersType tree 72060793789068864 = DETACHMENT LEADER
   should have 5 PersType children but has 4

Non-clustered PersType Counts

1. CIV_POST_TYPE 72060759428875115 = Head of State
   should have 3 PersType alignments but has 2
2. MIL_POST_TYPE 72060793789280990 = COMMANDER
   should have 5 PersType alignments but has 2

Figure 54. Web browser output of validation using business rules.

5. Analysis

5.1 XSD and XSLT

The use of XML Schema to validate an XML data fil is a common practice. Definin rules to test referential integrity is a logical extension and is within the capabilities of XSD. Going beyond the canonical form to describe the hierarchical data schema as a hierarchical XML schema provides XSD with the ability to perform additional validations.

A drawback is the amount of time required to validate data (*20*). While no benchmarking has been performed, because optimization of the XSD is beyond the scope of the project, very large file take a considerable amount of time to be validated. I intend to write a modifie version of GFMIEDM341.xsd that ignores uniqueness and referential integrity.[27] The new XSD will be limited to basic schema tests such as verifying the element type and value. Once a GFM XML data fil has been validated to ensure that element names are spelled correctly, no mandatory field are missing, etc., the more thorough XSD validation fil may be used.

The same reasoning applies to the XSLT scripts. XSLT transformation engines load the entire data fil into memory at once, and this is not feasible for huge data files The scripts make multiple passes through the XML data tree (not the file) once for each test. This should not have a major impact on the time required because the data is cached; however, if certain tests take too long to run, they may be removed from the main template near the top of the script.

The firs validation script should not be run until a given GFM XML data fil has been validated against the XSD. Likewise, the second (business rules) script should be run only after all issues discovered by the firs script are resolved. Additional scripts may be written to enforce business rules that are not shared by all users of the GFM data.

5.2 XML Limitations

GFM data is complex because it is both temporal and dependent on other data elements. A complete GFM data set may contain multiple force structure trees based on time and link type. The date/time groups that are tested in section 3.3.4 are used to designate when each object is active. A simple example is shown in figur 55.

If node A's active period of time does not overlap the time interval for node B, then it is impossible for A to ever be the parent of B. The naive rule used by the GFM validation, described in section 3.3.1, completely ignores the DTG elements. The template would see that the link's endpoints, namely A and B, exist, and decide that the link is valid.

[27] This fil may also allow fragmentary data to be validated.

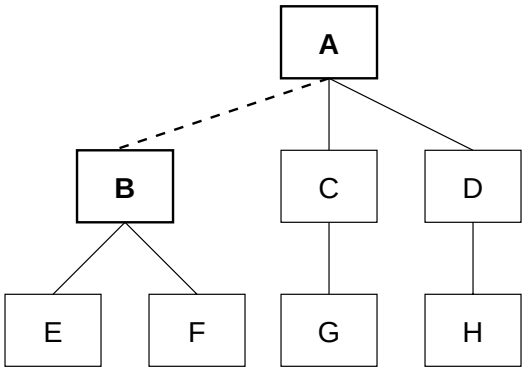

Figure 55. Sample force structure (organisation) tree.

The heuristics of the code required to definit vely validate the link may be too complex to implement in any language. If the nodes and link defin three different DTG intervals that partially overlap, is the link valid? The matter may require human judgment to confir that the intervals properly defin the actual data being modeled.

The validation of a tree based on link types may be performed only if the tree is traversed from the top down. The force structure tree that is produced from a given set of data—ignoring the time intervals for the moment—depends on the desired set of link types. When DTG intervals are included, the construction of the tree becomes much more complex. In figur 55, if the A–B link is broken, either because of the DTG or link type, then not only does node B disappear, but so do nodes E and F.

To summarize, GFM validation is performed at the link level and disregards DTG intervals (except as described in section 3.3.4). It does not perform tree validation.

5.3 Alternatives

Two big advantages of XSD and XSLT are that they are portable, and tools that may perform validations are readily available. GFM XML data may be validated on a server before it is distributed to a client that has requested the data. However, due to language limitations and inefficien y, it may be more practical to write one or more custom applications.

An alternative shortcut is to use SQL. Many commercial implementations of SQL include tests for referential integrity. It is a declarative language, as is XSLT, and the GFM XSLT scripts could easily be translated into SQL. Instead of parsing the GFM XML data into an XML data tree in memory, the fil would be incrementally parsed and its data stored in an SQL database. The nature of the tests requires that the data be scanned repeatedly, such as in the validation test “For each OBJ_TYPE, fin an OBJ_TYPE_ESTAB with the same value for its OBJ_TYPE_ID,” and repeated scans of the data cache may be extremely inefficient Performing queries on SQL data in a database is analogous to matching XSLT templates to an XML data tree. Due in part of the maturity of SQL, such queries may be highly optimized.

6. References

1. Chamberlain, S. C.; Boller, M.; Sprung, G.; Badami, V. Establishing a Community of Interest (COI) for Global Force Management. In *Proceedings of the 10th International Command and Control Research and Technology Symposium*; McLean, VA, 2005.

2. Extensible Markup Language (XML) 1.0. 2006 [ONLINE] Available http://www.w3.org/TR/2006/REC-xml-20060816/.

3. Deutsch, A.; Fernandez, M. F.; Florescu, D.; Levy, A. Y.; Maier, D.; Suciu, D. Querying XML Data. *IEEE Data Engineering Bulletin* **1999,** *22 (3),* 10–18.

4. XML Schema Part 1: Structures. 2004 [ONLINE] Available http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

5. XML Schema Part 2: Datatypes. 2004 [ONLINE] Available http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

6. Harold, E. R.; Means, S. W. *XML in a Nutshell, Third Edition;* O'Reilly Media, Inc.: Sebastopol, CA, 2004.

7. JC3IEDM Browse Representation. 2007 [ONLINE] Available http://www.mip-site.org/publicsite/04-Baseline_3.0/JC3IEDM-Joint_C3_Information_Exchange_Data_Model/HTML-Browser/index.html.

8. Chamberlain, S. C. Enterprise Identifier for Global Naming Across the C4I-Simulation Boundary. In *Proceedings of the 2001 Spring Simulation Interoperabiltiy Workshop*; Orlando, FL, 2001.

9. Dillon, S. XML to Relational: Bridging the gap. *Oracle Magazine* **2005,** *XIX* (5).

10. IC-ISM-v2. 2004 [ONLINE] Available http://www.niem.gov/IC-ISM-v2.xsd.

11. XML Path Language (XPath) Version 1.0. 1999 [ONLINE] Available http://www.w3.org/TR/1999/REC-xpath-19991116/.

12. XSL Transformations (XSLT) Version 1.0. 1999 [ONLINE] Available http://www.w3.org/TR/1999/REC-xslt-19991116/.

13. Kay, M. *XSLT 2.0 Programmer's Reference (Programmer to Programmer);* Wrox: Hoboken, NJ, 2004.

14. XHTML™1.0 The Extensible HyperText Markup Language. 2002 [ONLINE] Available http://www.w3.org/TR/2002/REC-xhtml1-20020801/.

15. Jelliffe, R. Using XSL as a Validation Language. 1999 [ONLINE] Available http://xml.ascc.net/en/utf-8/XSLvalidation.html.

16. Xerces Java Parser Readme. 2005 [ONLINE] Available http://xerces.apache.org/xerces-j/.

17. *XMLSpy*®, 2008 [ONLINE] Available http://www.altova.com/products/xmlspy/xml_editor.html.

18. Kay, M. *Saxon-B 9.0.0.4J*, 2008 [ONLINE] Available http://www.saxonica.com/.

19. Brundick, F. S.; Hartwig, Jr., G. W.; Chamberlain, S. C. *IChart: A Graphical Tool To View and Manipulate Force Management Structure Databases*; ARL-TR-4610; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, September 2008.

20. Nicola, M.; John, J. XML Parsing: A Threat to Database Performance. In *Proceedings of ACM International Conference on Information and Knowledge Management*; 2003.

Appendix A. Security Markings

The GFM XSD uses the security marking attributes define in the Intelligence Community Information Security Marking (IC-ISM) XSD file Data is classifie at the "record" level; the classificatio of the record is the highest classificatio of its "field elements.

The fil GFMIEDM341relatTableTypes.xsd imports the fil IC-ISM.xsd. The 'import' element is the mechanism that XSD uses to allow schemas to be shared and reused. The namespace for IC-ISM is "ism" and sample data is shown in figur A-1.

```
<OBJ_ITEM_ASSOC_TBL>
  <OBJ_ITEM_ASSOC ism:classification="U" ism:ownerProducer="USA"
                  ism:disseminationControls="FOUO">
    <SUBJ_OBJ_ITEM_ID>72060755133858488</SUBJ_OBJ_ITEM_ID>
    <OBJ_OBJ_ITEM_ID>72060755133863257</OBJ_OBJ_ITEM_ID>
    <OBJ_ITEM_ASSOC_IX>72060755133858493</OBJ_ITEM_ASSOC_IX>
    <CAT_CODE>HSADMI</CAT_CODE>
    <SUBCAT_CODE>ALTFOR</SUBCAT_CODE>
    <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
    <GFM_SUBCAT_CODE>DEFALT</GFM_SUBCAT_CODE>
    <GFM_OBJ_ITEM_ASSOC_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_S_DTG>
    <GFM_OBJ_ITEM_ASSOC_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_T_DTG>
  </OBJ_ITEM_ASSOC>
  <OBJ_ITEM_ASSOC ism:classification="U" ism:ownerProducer="USA"
                  ism:disseminationControls="FOUO">
    <SUBJ_OBJ_ITEM_ID>72060755133858488</SUBJ_OBJ_ITEM_ID>
    <OBJ_OBJ_ITEM_ID>72060755133863255</OBJ_OBJ_ITEM_ID>
    <OBJ_ITEM_ASSOC_IX>72060755133858494</OBJ_ITEM_ASSOC_IX>
    <CAT_CODE>HSADMI</CAT_CODE>
    <SUBCAT_CODE>ALTFOR</SUBCAT_CODE>
    <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
    <GFM_SUBCAT_CODE>DEFALT</GFM_SUBCAT_CODE>
    <GFM_OBJ_ITEM_ASSOC_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_S_DTG>
    <GFM_OBJ_ITEM_ASSOC_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_T_DTG>
  </OBJ_ITEM_ASSOC>
</OBJ_ITEM_ASSOC_TBL>
```

Figure A-1. GFM XML data with security attributes.

The IC-ISM XSD define several attributes, the details of which are not pertinent to this discussion. Two attribute groups, which are similar to the element group shown in figur 9, are define to allow the XSD writer to refer to an entire set of attributes. The **SecurityAttributesGroup** group states that the firs two attributes are mandatory and the rest are optional, while **SecurityAttributesOptionGroup** declares each attribute to be optional.

The GFM XSD designers wanted to maintain compatibility with data that had been created before security markings were introduced in the schema. A pair of new attribute groups are define in

the GFM namespace with each containing a different IC-ISM attribute group. The complete code is shown in figur A-2.

```
<!-- This group has mandatory attributes. -->
  <xs:attributeGroup name="SecurityAttributesGroup">
    <xs:annotation>
      <xs:documentation xml:lang="en">
The group of Information Security Marking attributes in which
the use of attributes 'classification' and 'ownerProducer' is
required. This group is to be contrasted with group
'SecurityAttributesOptionGroup' in which use of those attributes
is optional.
      </xs:documentation>
    </xs:annotation>
    <xs:attributeGroup ref="ism:SecurityAttributesGroup"/>
  </xs:attributeGroup>

<!-- All attributes are optional in the second group. -->
  <xs:attributeGroup name="xxxSecurityAttributesGroup">
    <xs:annotation>
      <xs:documentation xml:lang="en">
The group of Information Security Marking attributes in which
the use of all attributes is optional.
      </xs:documentation>
    </xs:annotation>
    <xs:attributeGroup ref="ism:SecurityAttributesOptionGroup"/>
  </xs:attributeGroup>
```

Figure A-2. Classificatio markings attribute groups.

Notice that the group names are almost identical; the second has an "xxx" prefix The reason is that security markings are not part of the GFMIEDM but were added to the GFM XSD. The user may edit the GFMIEDM341relatTableTypes.xsd fil and move the "xxx" prefi from the second group name to the first Since the attribute group name of **SecurityAttributesGroup** is used throughout this XSD file this will have the effect of declaring the security attributes to be optional. The relevant part of figur 8 is shown in figur A-3 with the attribute group highlighted.

```
<xs:complexType name="ObjectItem">
  <xs:annotation>
    <xs:documentation>Definition: An individually identified object
    that has military or civilian significance.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    ...
  </xs:sequence>
  <xs:attributeGroup ref="SecurityAttributesGroup"/>
</xs:complexType>
```

Figure A-3. Object Item type with security attribute group.

Appendix B. Validation Constraints

B.1 Structural Validation

The fil GFMIEDM341validate.xsl performs the following structural tests:

1. In the generalization hierarchy, an element's CAT_CODE must match the type of its child element. This test is performed only for elements which are children of elements whose name ends with '_OO_TBL'.
2. Both endpoints of a link must exist. The exception is an OBJ_TYPE_ESTAB_OBJT_DET whose GFM_OTEOD_ROLE_IND_CD has the value `1`; it never has a child.
3. When a pair of elements have both JC3 and GFM enumerated values, the GFM values are extensions of the JC3 set. The implementation rule is "If the GFM fiel is `NOS` (no such), then use the value in the JC3 field else use the GFM field s value." In practice, this means that at least one of the two field must have the value `NOS` or the firs enumerated value if `NOS` is not an enumerated value.
4. A starting date/time group (DTG) must be before the corresponding termination DTG.
5. An effective date/time or modernization date/time must be within the time span in item 4. Expressed mathematically, $s\_dtg \leq date\_time < t\_dtg$.
6. Every OBJ_TYPE must have an OBJ_TYPE_ESTAB.
7. Every OBJ_ITEM must appear as the OBJ_ITEM_ID in an OBJ_ITEM_OBJ_TYPE_ESTAB.
8. Every OBJ_TYPE must be associated with the proper OBJ_ITEM type.
9. When an element references an OBJ_TYPE and OBJ_TYPE_ESTAB, the OBJ_TYPE_ESTAB must refer to the same OBJ_TYPE.
10. Each tree may have at most one root node.

B.2 Business Rules

The fil GFMIEDM341businessRules.xsl tests GFM XML data for conformance with these business rules, which are subject to future changes and extensions.

1. The values for the CAT_CODE and SUBCAT_CODE elements of a link depend on the class of the parent and child objects. Only certain combinations of parent and child objects are permitted. The allowable values are shown in tables B-1 and B-2.

2. When a PERS_TYPE has a CAT_CODE of `MILTRY`, the SUBCAT_CODE must be `NOS`.

3. When a PERS_TYPE has a CAT_CODE of `CIV`, the SUBCAT_CODE must be `GOVEMP` or `NONGVE`.

4. Every ORG must appear in the Organisation tree, under the assumption that every object that is created should be used (unless it is reference data). In other words, it must be the OBJ_OBJ_ITEM_ID of an OBJ_ITEM_ASSOC.

5. A GFM_BILLET must never have a child in the Organisation tree. It may never be the SUBJ_OBJ_ITEM_ID of an OBJ_ITEM_ASSOC.

6. The OBJ_TYPE_ESTAB of a MAT_TYPE or PERS_TYPE that has child objects must have a CAT_CODE of `PCG`.

7. Because a CREW_PLATFORM_TYPE carries people, there must be at least one MAT_TYPE aligned with each one.

8. Each CIV_POST_TYPE and MIL_POST_TYPE must have at least one PERS_TYPE aligned with it.

9. When a PERS_TYPE is the root of a tree, it must reference a GFM_PERS_TYPE-_SKILL_ATTR with a type code of `ROS`.

10. Each Military Person Type tree must have 5 nodes, while a Civilian tree has 3 nodes.

11. When clustering is not used, each MIL_POST_TYPE must have 5 Person Types aligned with it, while a CIV_POST_TYPE must have 3 Person Types.

12. There should normally be GFM_CREW_PLATFORMs and GFM_BILLETs in an XML file

Table B-1. OTEOD category codes.

Parent	Child	CAT	SUBCAT	GFM_CAT	GFM_SUBCAT
ORG_TYPE	ORG_TYPE	HSADMI	ALTFOR	NOS	DEFALT
ORG_TYPE	ORG_TYPE	NOS	ALTFOR	COCOM	ASSIGN
ORG_TYPE	ORG_TYPE	NOS	ALTFOR	COCOM	UNASGN
ORG_TYPE	ORG_TYPE	CMDCTL	ALTFOR	NOS	DEFALT
ORG_TYPE	ORG_TYPE	CMDCTL	OPCON	NOS	NOS
ORG_TYPE	MAT_TYPE	ISAUTO	ALTFOR	NOS	DEFALT
ORG_TYPE	PERS_TYPE	ISAUTO	ALTFOR	NOS	DEFALT
MAT_TYPE	MAT_TYPE	ISPART	ALTFOR	NOS	DEFALT
PERS_TYPE	PERS_TYPE	ISPART	ALTFOR	NOS	DEFALT

Table B-2. Assoc category codes.

Parent	Child	CAT	SUBCAT	GFM_CAT	GFM_SUBCAT
ORG	ORG	HSADMI	ALTFOR	NOS	DEFALT
ORG	ORG	NOS	ALTFOR	COCOM	ASSIGN
ORG	ORG	NOS	ALTFOR	COCOM	UNASGN
ORG	ORG	CMDCTL	ALTFOR	NOS	DEFALT
ORG	ORG	CMDCTL	OPCON	NOS	NOS

INTENTIONALLY LEFT BLANK.

Appendix C. Valid Example

The sample GFM XML data fil shown in this set of figure demonstrates most of the structural and business rules. In order to save space, all mandatory elements that are not germane to the validation have been deleted. The fil uses the standard XML wrapper as shown in figur 49. Primary FMIDSs are shown boxed and foreign keys are in *italics*. Category codes and matching child elements are shown in **bold**. A summary of the data elements is after the figures

```
<OBJ_TYPE_OO_TBL>
<OBJ_TYPE ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <OBJ_TYPE_ID>72337338142818314</OBJ_TYPE_ID>
  <CAT_CODE>OR</CAT_CODE>
  <GFM_OBJ_TYPE_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_S_DTG>
  <GFM_OBJ_TYPE_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_T_DTG>
  <ORG_TYPE>
    <ORG_TYPE_ID>72337338142818314</ORG_TYPE_ID>
    <CAT_CODE>GVTORG</CAT_CODE>
    <GOVT_ORG_TYPE>
      <GOVT_ORG_TYPE_ID>72337338142818314</GOVT_ORG_TYPE_ID>
      <CAT_CODE>MILORG</CAT_CODE>
      <MIL_ORG_TYPE>
        <MIL_ORG_TYPE_ID>72337338142818314</MIL_ORG_TYPE_ID>
        <CAT_CODE>NOS</CAT_CODE>
        <GFM_CAT_CODE>CREW</GFM_CAT_CODE>
        <GFM_CREW_PLATFORM_TYPE>
          <GFM_CREW_PLATFORM_TYPE_ID>72337338142818314</GFM_CREW_PLATFORM_TYPE_ID>
        </GFM_CREW_PLATFORM_TYPE>
      </MIL_ORG_TYPE>
    </GOVT_ORG_TYPE>
  </ORG_TYPE>
</OBJ_TYPE>

<OBJ_TYPE ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <OBJ_TYPE_ID>72337338142818317</OBJ_TYPE_ID>
  <CAT_CODE>OR</CAT_CODE>
  <GFM_OBJ_TYPE_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_S_DTG>
  <GFM_OBJ_TYPE_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_T_DTG>
  <ORG_TYPE>
    <ORG_TYPE_ID>72337338142818317</ORG_TYPE_ID>
    <CAT_CODE>GVTORG</CAT_CODE>
    <GOVT_ORG_TYPE>
      <GOVT_ORG_TYPE_ID>72337338142818317</GOVT_ORG_TYPE_ID>
      <CAT_CODE>MILORG</CAT_CODE>
      <MIL_ORG_TYPE>
        <MIL_ORG_TYPE_ID>72337338142818317</MIL_ORG_TYPE_ID>
        <CAT_CODE>MILPST</CAT_CODE>
        <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
        <MIL_POST_TYPE>
          <MIL_POST_TYPE_ID>72337338142818317</MIL_POST_TYPE_ID>
        </MIL_POST_TYPE>
      </MIL_ORG_TYPE>
    </GOVT_ORG_TYPE>
  </ORG_TYPE>
</OBJ_TYPE>
...
```

Figure C-1. OBJ_TYPE elements, part 1 of 2 (ORG_TYPEs).

```
...
<OBJ_TYPE ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <OBJ_TYPE_ID> 72057594037927968 </OBJ_TYPE_ID>
  <CAT_CODE>MA</CAT_CODE>
  <GFM_OBJ_TYPE_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_S_DTG>
  <GFM_OBJ_TYPE_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_T_DTG>
  <MAT_TYPE>
    <MAT_TYPE_ID>72057594037927968</MAT_TYPE_ID>
    <CAT_CODE>EQ</CAT_CODE>
    <EQPT_TYPE>
      <EQPT_TYPE_ID>72057594037927968</EQPT_TYPE_ID>
      <CAT_CODE>VEHCLE</CAT_CODE>
      <VEHICLE_TYPE>
        <VEHICLE_TYPE_ID>72057594037927968</VEHICLE_TYPE_ID>
      </VEHICLE_TYPE>
    </EQPT_TYPE>
  </MAT_TYPE>
</OBJ_TYPE>

<OBJ_TYPE ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <OBJ_TYPE_ID> 72059647032297831 </OBJ_TYPE_ID>
  <CAT_CODE>PE</CAT_CODE>
  <GFM_OBJ_TYPE_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_S_DTG>
  <GFM_OBJ_TYPE_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_T_DTG>
  <PERS_TYPE>
    <PERS_TYPE_ID>72059647032297831</PERS_TYPE_ID>
    <CAT_CODE>MILTRY</CAT_CODE>
    <SUBCAT_CODE>NOS</SUBCAT_CODE>
    <GFM_PERS_TYPE_SKILL_ATTS>72059647032297828</GFM_PERS_TYPE_SKILL_ATTS>
  </PERS_TYPE>
</OBJ_TYPE>
</OBJ_TYPE_OO_TBL>
```

Figure C-2. OBJ_TYPE elements, part 2 of 2 (MAT_TYPE and PERS_TYPE).

```
<OBJ_TYPE_ESTAB_TBL>
<OBJ_TYPE_ESTAB ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <ESTABD_OBJ_TYPE_ID>72337338142818314</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818315</OBJ_TYPE_ESTAB_IX>
  <EFFCTV_DTTM>19900101000000.000</EFFCTV_DTTM>
  <GFM_OBJ_TYPE_ESTAB_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_S_DTG>
  <GFM_OBJ_TYPE_ESTAB_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_T_DTG>
</OBJ_TYPE_ESTAB>

<OBJ_TYPE_ESTAB ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <ESTABD_OBJ_TYPE_ID>72337338142818317</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818318</OBJ_TYPE_ESTAB_IX>
  <EFFCTV_DTTM>19900101000000.000</EFFCTV_DTTM>
  <GFM_OBJ_TYPE_ESTAB_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_S_DTG>
  <GFM_OBJ_TYPE_ESTAB_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_T_DTG>
</OBJ_TYPE_ESTAB>

<OBJ_TYPE_ESTAB ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <ESTABD_OBJ_TYPE_ID>72057594037927968</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72057594037928968</OBJ_TYPE_ESTAB_IX>
  <EFFCTV_DTTM>19900101000000.000</EFFCTV_DTTM>
  <GFM_OBJ_TYPE_ESTAB_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_S_DTG>
  <GFM_OBJ_TYPE_ESTAB_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_T_DTG>
</OBJ_TYPE_ESTAB>

<OBJ_TYPE_ESTAB ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <ESTABD_OBJ_TYPE_ID>72059647032297831</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72059647032297832</OBJ_TYPE_ESTAB_IX>
  <EFFCTV_DTTM>19900101000000.000</EFFCTV_DTTM>
  <CAT_CODE>PCG</CAT_CODE>
  <GFM_OBJ_TYPE_ESTAB_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_S_DTG>
  <GFM_OBJ_TYPE_ESTAB_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_TYPE_ESTAB_T_DTG>
</OBJ_TYPE_ESTAB>
</OBJ_TYPE_ESTAB_TBL>
```

Figure C-3. OBJ_TYPE_ESTAB elements.

```
<GFM_PERS_TYPE_SKILL_ATTR_TBL>
<GFM_PERS_TYPE_SKILL_ATTR ism:classification="U" ism:ownerProducer="USA" ...>
  <GFM_PERST_SKILL_ATTR_ID>72059647032297828</GFM_PERST_SKILL_ATTR_ID>
  <GFM_PERST_SKILL_ATTR_NAME_TXT>ROOT OCCUPATIONAL SPECIALTY</GFM_PERST_SKILL_ATTR_NAME_TXT>
  <GFM_PERST_SKILL_ATTR_TYPE_CD>ROS</GFM_PERST_SKILL_ATTR_TYPE_CD>
  <GFM_PERST_SKILL_ATTR_OWNER_CD>USA</GFM_PERST_SKILL_ATTR_OWNER_CD>
  <GFM_PERST_SKILL_ATTR_CAT>OFFICER</GFM_PERST_SKILL_ATTR_CAT>
  <GFM_PERST_SKILL_ATTR_S_DTG>1990-01-01T00:00:00Z</GFM_PERST_SKILL_ATTR_S_DTG>
  <GFM_PERST_SKILL_ATTR_T_DTG>2999-12-01T00:00:00Z</GFM_PERST_SKILL_ATTR_T_DTG>
</GFM_PERS_TYPE_SKILL_ATTR>
</GFM_PERS_TYPE_SKILL_ATTR_TBL>
```

Figure C-4. GFM_PERS_TYPE_SKILL_ATTR element.

```
<OBJ_TYPE_ESTAB_OBJT_DET_TBL>
<OBJ_TYPE_ESTAB_OBJT_DET ism:classification="U" ism:ownerProducer="USA" ...>
  <ESTABD_OBJ_TYPE_ID>72337338142818314</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818315</OBJ_TYPE_ESTAB_IX>
  <OBJ_TYPE_ESTAB_OBJT_DET_IX>72337338142818321</OBJ_TYPE_ESTAB_OBJT_DET_IX>
  <DET_OBJ_TYPE_ID>72337338142818317</DET_OBJ_TYPE_ID>
  <DET_OBJ_TYPE_ESTAB_IX>72337338142818318</DET_OBJ_TYPE_ESTAB_IX>
  <GFM_OTEOD_CAT_CODE>HSADMI</GFM_OTEOD_CAT_CODE>
  <GFM_OTEOD_SUBCAT_CODE>ALTFOR</GFM_OTEOD_SUBCAT_CODE>
  <GFM_OTEOD_GFM_CAT_CODE>NOS</GFM_OTEOD_GFM_CAT_CODE>
  <GFM_OTEOD_GFM_SUBCAT_CODE>DEFALT</GFM_OTEOD_GFM_SUBCAT_CODE>
  <GFM_OBJT_ESTAB_OBJT_DET_S_DTG>1990-01-01T00:00:00Z</GFM_OBJT_ESTAB_OBJT_DET_S_DTG>
  <GFM_OBJT_ESTAB_OBJT_DET_T_DTG>2999-12-01T00:00:00Z</GFM_OBJT_ESTAB_OBJT_DET_T_DTG>
</OBJ_TYPE_ESTAB_OBJT_DET>

<OBJ_TYPE_ESTAB_OBJT_DET ism:classification="U" ism:ownerProducer="USA" ...>
  <ESTABD_OBJ_TYPE_ID>72337338142818314</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818315</OBJ_TYPE_ESTAB_IX>
  <OBJ_TYPE_ESTAB_OBJT_DET_IX>72337338142818376</OBJ_TYPE_ESTAB_OBJT_DET_IX>
  <DET_OBJ_TYPE_ID>72057594037927968</DET_OBJ_TYPE_ID>
  <DET_OBJ_TYPE_ESTAB_IX>72057594037928968</DET_OBJ_TYPE_ESTAB_IX>
  <GFM_OTEOD_CAT_CODE>ISAUTO</GFM_OTEOD_CAT_CODE>
  <GFM_OTEOD_SUBCAT_CODE>ALTFOR</GFM_OTEOD_SUBCAT_CODE>
  <GFM_OTEOD_GFM_CAT_CODE>NOS</GFM_OTEOD_GFM_CAT_CODE>
  <GFM_OTEOD_GFM_SUBCAT_CODE>DEFALT</GFM_OTEOD_GFM_SUBCAT_CODE>
  <GFM_OBJT_ESTAB_OBJT_DET_S_DTG>1990-01-01T00:00:00Z</GFM_OBJT_ESTAB_OBJT_DET_S_DTG>
  <GFM_OBJT_ESTAB_OBJT_DET_T_DTG>2999-12-01T00:00:00Z</GFM_OBJT_ESTAB_OBJT_DET_T_DTG>
</OBJ_TYPE_ESTAB_OBJT_DET>

<OBJ_TYPE_ESTAB_OBJT_DET ism:classification="U" ism:ownerProducer="USA" ...>
  <ESTABD_OBJ_TYPE_ID>72337338142818317</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818318</OBJ_TYPE_ESTAB_IX>
  <OBJ_TYPE_ESTAB_OBJT_DET_IX>72337338142818380</OBJ_TYPE_ESTAB_OBJT_DET_IX>
  <DET_OBJ_TYPE_ID>72059647032297831</DET_OBJ_TYPE_ID>
  <DET_OBJ_TYPE_ESTAB_IX>72059647032297832</DET_OBJ_TYPE_ESTAB_IX>
  <GFM_OTEOD_CAT_CODE>ISAUTO</GFM_OTEOD_CAT_CODE>
  <GFM_OTEOD_SUBCAT_CODE>ALTFOR</GFM_OTEOD_SUBCAT_CODE>
  <GFM_OTEOD_GFM_CAT_CODE>NOS</GFM_OTEOD_GFM_CAT_CODE>
  <GFM_OTEOD_GFM_SUBCAT_CODE>DEFALT</GFM_OTEOD_GFM_SUBCAT_CODE>
  <GFM_OBJT_ESTAB_OBJT_DET_S_DTG>1990-01-01T00:00:00Z</GFM_OBJT_ESTAB_OBJT_DET_S_DTG>
  <GFM_OBJT_ESTAB_OBJT_DET_T_DTG>2999-12-01T00:00:00Z</GFM_OBJT_ESTAB_OBJT_DET_T_DTG>
</OBJ_TYPE_ESTAB_OBJT_DET>
</OBJ_TYPE_ESTAB_OBJT_DET_TBL>
```

Figure C-5. OBJ_TYPE_ESTAB_OBJT_DET (link) elements.

```
<OBJ_ITEM_OO_TBL>
<OBJ_ITEM ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <OBJ_ITEM_ID> 72337338142818342 </OBJ_ITEM_ID>
  <CAT_CODE>OR</CAT_CODE>
  <GFM_OBJ_ITEM_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_S_DTG>
  <GFM_OBJ_ITEM_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_T_DTG>
  <ORG>
    <ORG_ID>72337338142818342</ORG_ID>
    <CAT_CODE>NOS</CAT_CODE>
    <GFM_CAT_CODE>CR</GFM_CAT_CODE>
    <GFM_CREW_PLATFORM>
      <GFM_CREW_PLATFORM_ID>72337338142818342</GFM_CREW_PLATFORM_ID>
    </GFM_CREW_PLATFORM>
  </ORG>
</OBJ_ITEM>

<OBJ_ITEM ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <OBJ_ITEM_ID> 72337338142818344 </OBJ_ITEM_ID>
  <CAT_CODE>OR</CAT_CODE>
  <GFM_OBJ_ITEM_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_S_DTG>
  <GFM_OBJ_ITEM_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_T_DTG>
  <ORG>
    <ORG_ID>72337338142818344</ORG_ID>
    <CAT_CODE>NOS</CAT_CODE>
    <GFM_CAT_CODE>BL</GFM_CAT_CODE>
    <GFM_BILLET>
      <GFM_BILLET_ID>72337338142818344</GFM_BILLET_ID>
    </GFM_BILLET>
  </ORG>
</OBJ_ITEM>
</OBJ_ITEM_OO_TBL>
```

Figure C-6. OBJ_ITEM (ORG) elements.

```
<OBJ_ITEM_OBJ_TYPE_ESTAB_TBL>
<OBJ_ITEM_OBJ_TYPE_ESTAB ism:classification="U" ism:ownerProducer="USA" ...>
  <OBJ_ITEM_ID>72337338142818342</OBJ_ITEM_ID>
  <ESTABD_OBJ_TYPE_ID>72337338142818314</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818315</OBJ_TYPE_ESTAB_IX>
  <OBJ_ITEM_OBJ_TYPE_ESTAB_IX> 72337338142818343 </OBJ_ITEM_OBJ_TYPE_ESTAB_IX>
  <EFFCTV_DTTM>19900101000000.000</EFFCTV_DTTM>
  <GFM_ORG_ORGT_M_DTG>1990-01-01T00:00:00Z</GFM_ORG_ORGT_M_DTG>
  <GFM_OBJI_OBJT_ESTAB_S_DTG>1990-01-01T00:00:00Z</GFM_OBJI_OBJT_ESTAB_S_DTG>
  <GFM_OBJI_OBJT_ESTAB_T_DTG>2999-12-01T00:00:00Z</GFM_OBJI_OBJT_ESTAB_T_DTG>
</OBJ_ITEM_OBJ_TYPE_ESTAB>

<OBJ_ITEM_OBJ_TYPE_ESTAB ism:classification="U" ism:ownerProducer="USA" ...>
  <OBJ_ITEM_ID>72337338142818344</OBJ_ITEM_ID>
  <ESTABD_OBJ_TYPE_ID>72337338142818317</ESTABD_OBJ_TYPE_ID>
  <OBJ_TYPE_ESTAB_IX>72337338142818318</OBJ_TYPE_ESTAB_IX>
  <OBJ_ITEM_OBJ_TYPE_ESTAB_IX> 72337338142818345 </OBJ_ITEM_OBJ_TYPE_ESTAB_IX>
  <EFFCTV_DTTM>19900101000000.000</EFFCTV_DTTM>
  <GFM_ORG_ORGT_M_DTG>1990-01-01T00:00:00Z</GFM_ORG_ORGT_M_DTG>
  <GFM_OBJI_OBJT_ESTAB_S_DTG>1990-01-01T00:00:00Z</GFM_OBJI_OBJT_ESTAB_S_DTG>
  <GFM_OBJI_OBJT_ESTAB_T_DTG>2999-12-01T00:00:00Z</GFM_OBJI_OBJT_ESTAB_T_DTG>
</OBJ_ITEM_OBJ_TYPE_ESTAB>
</OBJ_ITEM_OBJ_TYPE_ESTAB_TBL>
```

Figure C-7. OBJ_ITEM_OBJ_TYPE_ESTAB elements.

```
<OBJ_ITEM_ASSOC_TBL>
<OBJ_ITEM_ASSOC ism:classification="U" ism:ownerProducer="USA" ism:disseminationControls="FOUO">
  <SUBJ_OBJ_ITEM_ID>72337338142818342</SUBJ_OBJ_ITEM_ID>
  <OBJ_OBJ_ITEM_ID>72337338142818344</OBJ_OBJ_ITEM_ID>
  <OBJ_ITEM_ASSOC_IX> 72337338142818352 </OBJ_ITEM_ASSOC_IX>
  <CAT_CODE>HSADMI</CAT_CODE>
  <SUBCAT_CODE>ALTFOR</SUBCAT_CODE>
  <GFM_CAT_CODE>NOS</GFM_CAT_CODE>
  <GFM_SUBCAT_CODE>DEFALT</GFM_SUBCAT_CODE>
  <GFM_OBJ_ITEM_ASSOC_S_DTG>1990-01-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_S_DTG>
  <GFM_OBJ_ITEM_ASSOC_T_DTG>2999-12-01T00:00:00Z</GFM_OBJ_ITEM_ASSOC_T_DTG>
</OBJ_ITEM_ASSOC>
</OBJ_ITEM_ASSOC_TBL>
```

Figure C-8. OBJ_ITEM_ASSOC element.

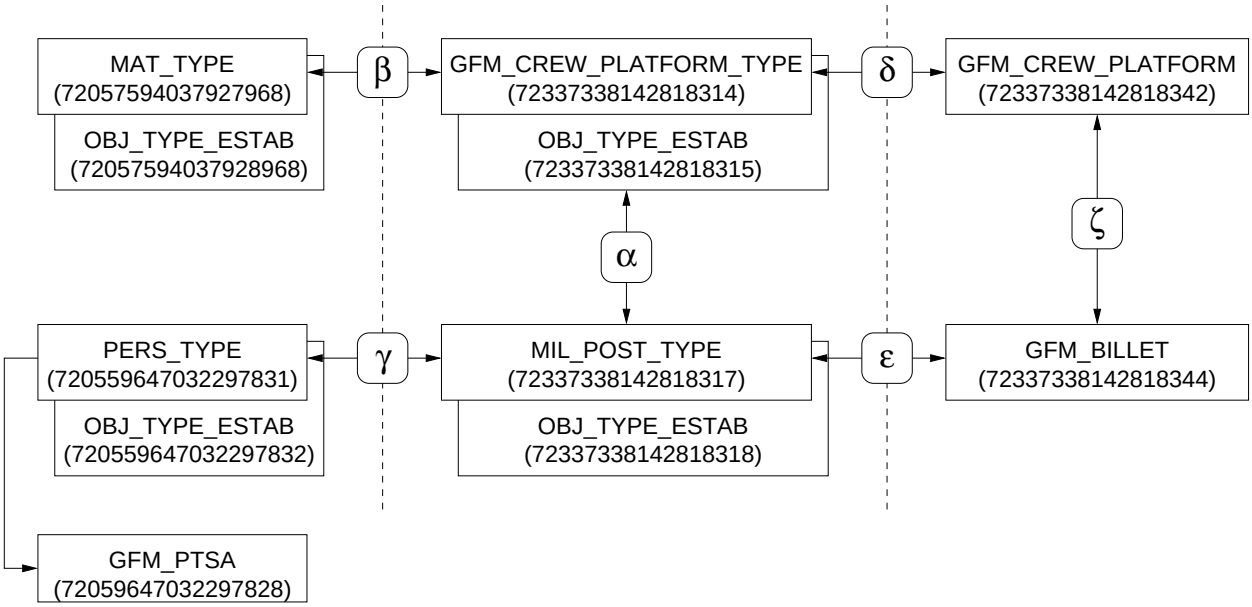

Figure C-9. Sample data with relationships.

Table C-1. Link keys for figur C-9.

Category	Symbol	EwID
OBJ_TYPE_ESTAB_OBJT_DET	α	72337338142818321
	β	72337338142818376
	γ	72337338142818380
OBJ_ITEM_OBJ_TYPE_ESTAB	δ	72337338142818343
	ϵ	72337338142818345
OBJ_ITEM_ASSOC	ζ	72337338142818352

Figure C-9 and table C-1 graphically show the relationships between the data elements in figure C-1–8. The objects in the center of figur C-9 comprise the Org Type tree, while the boxes beneath them are the establishments of the object types. The firs OBJ_TYPE is a GFM_CREW_PLAT- FORM_TYPE, and its subordinate OBJ_TYPE is a MIL_POST_TYPE. The OBJ_TYPE_ESTAB_- OBJT_DET (OTEOD) labeled α is the link that connects the OBJ_TYPEs in the tree.

The right third of the diagram represents the Organisation tree. There are two OBJ_ITEMs that are connected by the ζ OBJ_ITEM_ASSOC link. Every OBJ_ITEM must be an instantiation of an OBJ_TYPE. The OBJ_ITEM_OBJ_TYPE_ESTAB objects (δ and ϵ) associate each OBJ_ITEM with its respective OBJ_TYPE.

Reference data is shown in the leftmost third of the diagram. A MAT_TYPE and its establishment are aligned with the GFM_CREW_PLATFORM_TYPE, and a PERS_TYPE and establishment are aligned with the MIL_POST_TYPE object. These are examples where OTEODs do double-duty, since the objects β and γ are both OTEOD links. In addition, the PERS_TYPE refers to the GFM_PERS_TYPE_SKILL_ATTR (GFM_PTSA) to indicate what skill attribute the PERS_TYPE implements.

The sample shows a one-to-one relationship between OBJ_TYPEs and OBJ_TYPE_ESTABs. This is not a requirement for GFM, although the IChart application generally assumes that this is the case. ORG_TYPE to ORG_TYPE links, such as link α, are used to construct the optional Org Type tree. The GFM business rules require that an ORG to point to an ORG_TYPE and its establishment, but the ORG_TYPE does not need to be a node in the Org Type tree.

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

CPT	CREW_PLATFORM_TYPE
DI	Data Initiative
DMWG	Data Modelling Working Group
DTG	date/time group
E-R	Entity-Relationship
EwID	Enterprise-Wide Identifie
GFM	Global Force Management
HTML	Hypertext Markup Language
IC-ISM	Intelligence Community Information Security Marking
IEDM	Information Exchange Data Model
JC3	Joint Command, Control and Consultation
JC3IEDM	Joint Command, Control and Consultation Information Exchange Data Model
MIP	Multilateral Interoperability Programme
ORG	Organisation
OTEOD	OBJ_TYPE_ESTAB_OBJT_DET
PCG	Parts Catalogue
PTSA	Person Type Skill Attribute
PTT	PersType Tree
RDBMSs	relational database management systems
XML	Extensible Markup Language
XSD	XML Schema Definitio
XSLT	XML Stylesheet Language: Transformations

Glossary

attribute An XML element may have attributes which are of the form *name="value"*. The GFM XSD uses attributes for classificatio marking of data elements.

complex type An XML element that contains attributes and/or child elements. In SQL terms, a fiel is a simple type, while a record is a complex type because it contains multiple fields

DTG A date/time group in GFM is an instant in time specifie by both a date and a time. The format used by the GFM XSD is "yyyy-mm-ddThh:mm:ssZ". (The **EFFCTV_DTTM** fiel is a DTTM and not a DTG and uses a different format.)

declarative language A high-level programming language that describes a problem rather than definin a solution. XSLT and SQL are declarative languages.

element The basic building block of an XML data file Data values are contained within matching start and end elements.

E-R Diagram An Entity-Relationship Diagram is a graphical way of showing the interrelationships between entities in a database.

EwID An Enterprise-Wide Identifie is a surrogate key that is globally unique within the GFM community. They are used as primary and foreign keys in the GFM XSD.

force structure tree The command and control hierarchy that contains elements of the same type. In this tree, a child element is subordinate to its parent element.

Generalization Hierarchy GFM term used to refer to data elements that have a parent/child relationship in the object-oriented sense.

GFM Global Force Management is a Joint Staff and Offic of the Secretary of Defense initiative designed to standardize force structure representation, making it visible, accessible, and understandable across the Department of Defense.

JC3IEDM The Joint Command, Control and Consultation Information Exchange Data Model is the message exchange mechanism for the Multilateral Interoperability Programme (MIP).

key An XSLT element that is assigned the value of the element specifie by an XPath expression.

keyref An XSLT element that compares the value of the element specifie by an XPath expression with the value of a specifie key. A key/keyref pair is used to test referential integrity.

match template An XSLT template that is invoked when its XPath expression matches an element (or attribute) in an XML data file It may be controlled by assigning a numerical priority and/or a named mode.

named template An XSLT template that is invoked by using its name like in a procedural language.

namespace The context for related elements and attributes to group components of a single XML application together. This also disambiguates multiple elements with the same name but different meanings.

procedural language A high-level programming language that describes a series of computational steps to be carried out. The majority of popular languages, including C and Java, are procedural languages.

referential integrity Consistency between coupled tables which is usually enforced by the combination of a primary key and a foreign key. The keys are EwIDs in the GFM model.

simple type An XML type that does not have child elements or attributes. Other languages call this a *scalar* type. Examples are strings and numbers.

template The basic element in an XSLT script.

validation Every XML data fil must reference a schema definitio fil (XSD). The data fil is valid if all of its elements and attributes are declared in the XSD and it conforms to the rules define in the XSD.

well-formedness The basic syntax which all XML documents or data file must follow. Rules specify constraints such as "Every start element must have a matching end element."

XHTML Extensible Hypertext Markup Language is a markup language that is a reformulation of HTML but also conforms to XML syntax.

XML Extensible Markup Language is the syntax used when exchanging GFM data between systems. Many of the specification developed by the World Wide Web Consortium (W3C) are written in XML.

XML data tree An XML data fil is processed by an XML parser (reader) and stored in memory in the form of a tree. Operations, such as template matching in XSLT, are performed on this memory-resident tree.

XPath A language for identifying particular parts of an XML document or data file It is tightly coupled with XSLT but is not written in XML.

XSD An XML Schema Definitio define elements, their types, their relationships to other types, and simple constraints. It is written in XML and is similar to a data dictionary.

XSLT XML Stylesheet Language: Transformations is an XML-based language to convert an XML fil into another form. A transformation engine reads the data fil and applies the templates define in the XSLT file (XSLT used to be called simply XSL until it was split into two parts.)

INTENTIONALLY LEFT BLANK.

No. of Copies	Organization
1 ELEC	ADMNSTR DEFNS TECHL INFO CTR ATTN DTIC OCP 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	DARPA ATTN IXO S WELBY 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1 CD	OFC OF THE SECY OF DEFNS ATTN ODDRE (R&AT) THE PENTAGON WASHINGTON DC 20301-3080
1	US ARMY RSRCH DEV AND ENGRG CMND ARMAMENT RSRCH DEV AND ENGRG CTR ARMAMENT ENGRG AND TECHNLGY CTR ATTN AMSRD AAR AEF T J MATTS BLDG 305 ABERDEEN PROVING GROUND MD 21005-5001
1 CD	JOINT STAFF J-8 MASO ATTN G SPRUNG ROOM 2C646 JOINT STAFF PENTAGON WASHINGTON DC 20318-8000
1	PM TIMS, PROFILER (MMS-P) AN/TMQ-52 ATTN B GRIFFIES BUILDING 563 FT MONMOUTH NJ 07703
1	US ARMY INFO SYS ENGRG CMND ATTN AMSEL IE TD F JENIA FT HUACHUCA AZ 85613-5300
1	COMMANDER US ARMY RDECOM ATTN AMSRD AMR W C MCCORKLE 5400 FOWLER RD REDSTONE ARSENAL AL 35898-5000

No. of Copies	Organization
1	US GOVERNMENT PRINT OFF DEPOSITORY RECEIVING SECTION ATTN MAIL STOP IDAD J TATE 732 NORTH CAPITOL ST NW WASHINGTON DC 20402
1	CFLCC PARC ATTN B PARRISH BLDG 505 FPO AE 09306
1	U.S. ARMY RSRCH LAB ATTN AMSRD ARL CI IC M MITTRICK BLDG 321 ABERDEEN PROVING GROUND MD 21005
7	US ARMY RSRCH LAB ATTN AMSRD ARL CI IC F S BRUNDICK (6 COPIES) ATTN AMSRD ARL CI IC G MOSS BLDG 321 ABERDEEN PROVING GROUND MD 21005
1	US ARMY RSRCH LAB ATTN AMSRD ARL CI IC M THOMAS BLDG 321 RM 1B ABERDEEN PROVING GROUND MD 21005
1	US ARMY RSRCH LAB ATTN AMSRD ARL CI IC S CHAMBERLAIN BLDG 321 ABERDEEN PROVING GROUND MD 21005
1	US ARMY RSRCH LAB ATTN AMSRD ARL CI OK TP TECHL LIB T LANDFRIED BLDG 4600 ABERDEEN PROVING GROUND MD 21005-5066

No. of Copies	Organization
1	DIRECTOR US ARMY RSRCH LAB ATTN AMSRD ARL RO EV W D BACH PO BOX 12211 RESEARCH TRIANGLE PARK NC 27709
5	US ARMY RSRCH LAB ATTN AMSRD ARL CI I B BROOME ATTN AMSRD ARL CI J GOWENS ATTN AMSRD ARL CI OK PE TECHL PUB ATTN AMSRD ARL CI OK TL TECHL LIB ATTN IMNE ALC HR MAIL & RECORDS MGMT ADELPHI MD 20783-1197

TOTAL: 27 (1 ELEC, 2 CDS, 24 HCS)